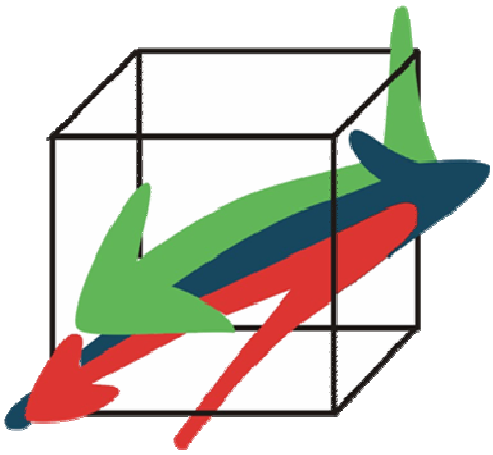
GUI-Anwendung zur Administration der fli4l-Installation



Systemhandbuch

Inhaltsverzeichnis

Einleitung 5
Teil 1 – Was ist fli4l? 6
1. Einführung 7
1.1. Allgemeines 7
1.2. Anforderungen und unterstützte Hardware/Protokolle 7
1.3. Installation 7
1.4. Konfiguration und Steuerung des fli4l 7
Teil 2 – Grundlagen der Entwicklung 8
Übersicht 9
2. Router 9
2.1. Einführung 9
2.2. Konzept 9
2.3. Aufbau der Infrastruktur 11
2.4. Handbuch 22
3. Infrastruktur 27
3.1. Einführung 27
3.2. Grundlagen 27
3.3. Einrichtung und Verwendung der Werkzeuge in Eclipse 31
3.4. Aufbau und Funktion des Ant-Skripts 37
3.5. Einrichtung von Jenkins 38
3.6. Nutzung von Jenkins 39
Teil 3 – Modelle und Funktionalitäten 43
Übersicht 44
4. Klassenmodelle 44
4.1. Einführung 44
4.2. Klassenmodell - Gesamt 45
4.3. Klassenmodell – Scanner 46
4.4. Klassenmodell – Parser 46
4.5. Klassenmodell - Printer 47
4.6. Klassenmodell – Pakete 48
4.7. Klassenmodell - GUI 49
5. Konfiguration 49
5.1. Einführung 49

5.2. Paketstruktur 49
5.3. Dateistruktur 50
5.4. Konfigurationsdateien 50
5.5. Fachlicher Entwurf 53
5.6. Technischer Entwurf 53
5.7. Eclipse/Java- Projektstruktur 57
6. Scanner 58
6.1. Einführung 58
6.2. Grundlagen 58
6.3. Analyse 58
6.4. Entwurf 59
6.5. Umsetzung 69
6.6. Fazit 69
7. Parser 70
7.1. Einführung 70
7.2. Analyse 70
7.3. Entwurf und Umsetzung 74
8. Printer 81
8.1. Einführung 81
8.2. Analyse 81
8.3. Entwurf 83
8.4. Implementierung 85
9. Pakete 86
9.1. Einführung 86
9.2. Grundlagen 86
9.3. Modellerweiterung 88
9.4. Scanner 90
9.5. Parser 101
9.6. Zusammenfassung 113
10. Grafische Benutzeroberfläche - GUI 114
10.1. Einführung 114
10.2. Analyse 114
10.3. Entwurf 117
10.4. Implementierung 122
10.5. Funktionen 127

10.6. Ausblick 130
Abbildungsverzeichnis 132
Tabellenverzeichnis 135
Glossar 136
11. Java-Dokumentation 138

Einleitung

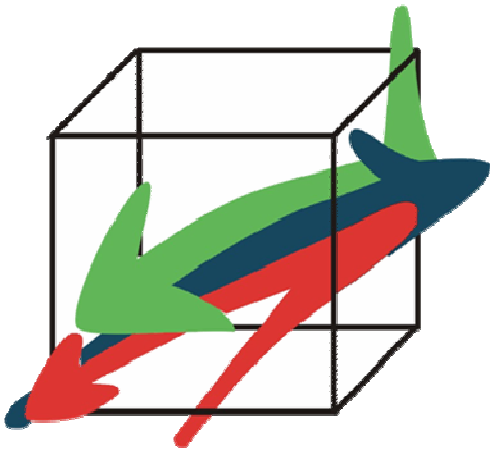
Dieses Handbuch soll den Anwender in die grafische Benutzeroberfläche (GUI – Grafical User Interface) von fli4l, eine Linux-Distribution, die einen Software-Router zur Verfügung stellt, einführen. Behandelt werden die Grundlagen der Entwicklung der Oberfläche, die zugehörigen Klassenmodelle sowie detaillierte Informationen zu Aufbau der Paketstruktur und Ablauf der Funktionalitäten.

Die Zielgruppe dieses Handbuchs sind Personen, die sich mit der Konfiguration und Administration des fli4l-Routers befassen. Ein grundsätzliches Verständnis für die Softwareentwicklung und -bedienung wird daher vorausgesetzt. Alle Elemente der grafischen Benutzeroberfläche sind in der Programmiersprache Java verfasst.

Das Handbuch befasst sich zum Zeitpunkt seiner Veröffentlichung mit der aktuellen Version 3.6.2 der fli4l- Software. Es ist daher nicht auszuschließen, dass die Bedienung der Software in zukünftigen Versionen abweichen kann.

Es sei daher empfohlen, zu jeder Version der Software das offizielle Benutzerhandbuch von fli4l zu studieren.

Teil 1 – Was ist fli4l?



1. Einführung

1.1. Allgemeines

fli4l (flexible internet router for linux, früher floppy isdn for linux) ist eine Linux-Distribution, die einen Software-Router zur Verfügung stellt. Dieser Router kann zwischen Ethernet und ISDN, UMTS und DSL bzw. zwischen zwei Ethernet-Netzwerken routen. [1]

1.2. Anforderungen und unterstützte Hardware/Protokolle

fli4l basiert auf dem Linux-Kernel und weist folgende Hardwareanforderungen auf: [2]

1. 486er CPU ab DX2/66, besser 486er DX4/100 oder Pentium ab 75 MHz

2. 12 MiB Speicher, besser 16 MiB

3. Ethernet-Netzwerkkarte(n)

4. ISDN: HiSax unterstützte ISDN-Karte (Typ 1-39) oder AVM ISA / PCI / USB

5. mindestens ein Diskettenlaufwerk

1.3. Installation

Informationen zur Installation der fli4l-Software sind der Anwenderdokumentation [3] zu entnehmen.

1.4. Konfiguration und Steuerung des fli4l

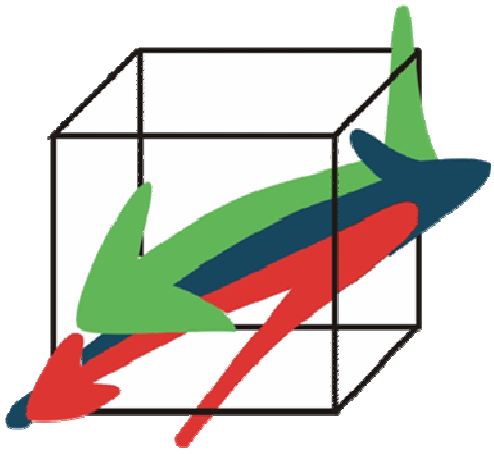
fli4l kann in seiner ursprünglichen Version unter Linux, Unix und Windows über Textdateien konfiguriert werden. Diese Möglichkeit der Konfiguration wird in diesem Handbuch um eine grafische Oberfläche erweitert. Dazu gehören eine versionierte Verwaltung von Konfigurationen, das Behandeln von Versionsaktualisierungen sowie die Synchronisation von Konfiguration und Programmpaketen mit den verwalteten Routern.

Ein fli4l-Router wird mittels Browser über eine Webschnittstelle gesteuert. Eine weitere Alternative ist das Programm ImonC (ISDN Monitor Client) für Windows und für Linux (GTK), das ein ferngesteuertes Update der Routersoftware ermöglicht. 4

ein Update kann auchvia SCP erfolg

[1] http://www.linuxwiki.de/fli4l
[2] http://www.fli4l.de/home/eigenschaften/funktionsumfang
[3] http://www.fli4l.de/hilfe/dokumentation/dokumentation-36x
[4] http://www.linuxwiki.de/fli4l

Teil 2 – Grundlagen der Entwicklung



Übersicht

Zum Verständnis der Funktionsweise des fli4l-Routers sind zunächst die Grundlagen des Entwicklungsprozesses als tragendes Fundament der Konfigurationsverwaltung wichtig.

In Kapitel 2 wird zuerst eine virtuelle Testumgebung mit mehreren Testinstallationen für den fli4l-Router vorgestellt, in der unterschiedliche Konfigurationen durchgeführt werden können.

Kapitel 3 widmet sich der Infrastruktur, die die Konfiguration durch einen Client ermöglicht. Dazu wurde ein Build-Script und Tools für Analysen und Aufbereitung von Skriptereignissen zur Verfügung gestellt.

2. Router

2.1. Einführung

In diesem Kapitel wird die Testumgebung für die fli4l-Konfigurationen behandelt. Dazu wurde zunächst die Infrastruktur einer virtuellen Testumgebung aufgestellt, die ein Netzwerk mit Routern und Clients simuliert. Diese Testumgebung wurde zur Durchführung der Testfälle auf mehreren Rechnern eingerichtet.

Innerhalb dieser Testumgebung wurde die fli4l-Software auf den virtuellen Routern installiert und mit unterschiedlichen Einstellungen konfiguriert.

2.2. Konzept

2.2.1. Infrastruktur

Die Testumgebung für das Integrationsprojekt wird für den Überblick in Abbildung 1 dargestellt.

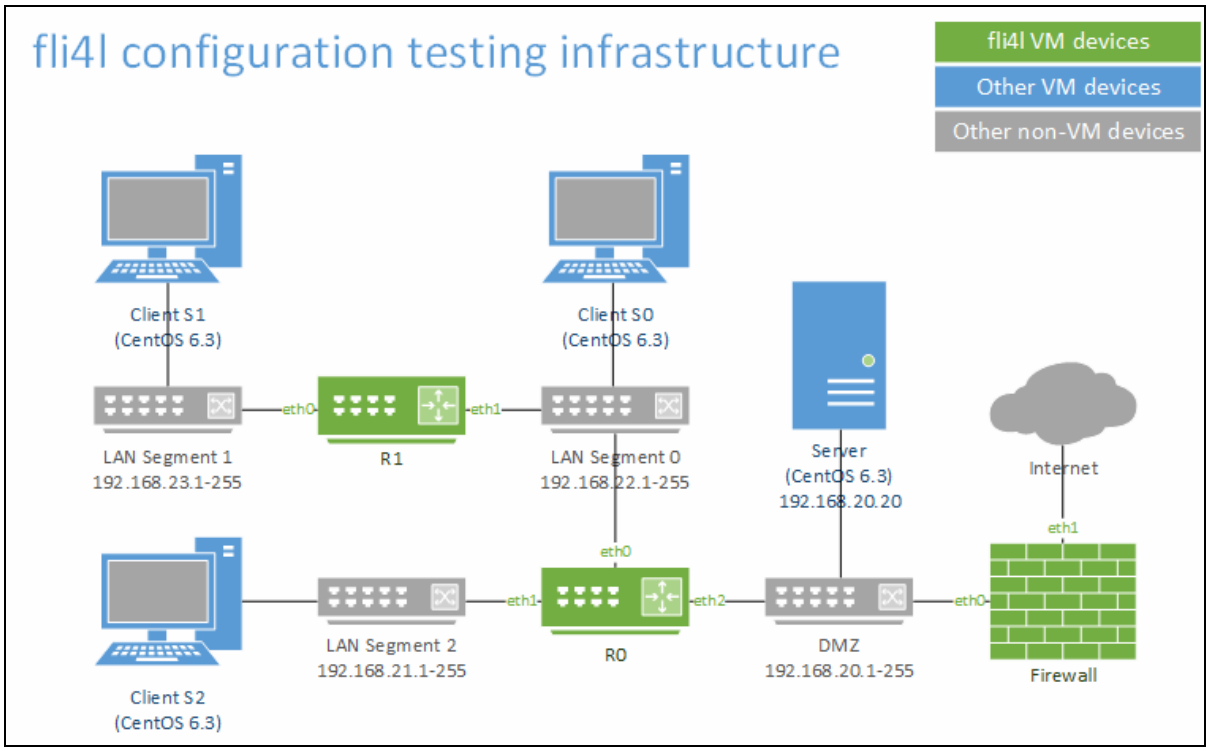


Abbildung 1: Testumgebung für das Integrationsprojekt

Innerhalb der Testumgebung gibt es vier verschiedene Netzwerke, die durch fli4l-Router verbunden werden. Jedes Netzwerk wird mithilfe eines LAN-Segmentes repräsentiert. Mit dem Router „Firewall“ ist es möglich eine Verbindung zum Internet herzustellen. In jedem Netzwerk befindet sich jeweils ein Client, um Verbindungstests aus jedem Netzwerk fahren zu können.

2.2.2. Netzwerke

Jedes Netzwerk der Testumgebung erhält seinen eigenen IP-Adressbereich (siehe Tabelle 1). Des Weiteren erhält jedes Netzwerk einen lokalen DHCP-Server, sowie einen lokalen DNS-Server. Der IP-Adressbereich eines jeden Netzwerkes, der von dem DHCP-Server mit IP-Adressen versorgt wird, ist der Bereich mit den Adressen 192.168.xxx.100-150. Einzige Ausnahme ist das Netzwerk „DMZ“, in welchem der DHCP-Server nur eine statische Vergabe der IP-Adresse anhand der MAC-Adresse für den Server bereitstellt (192.168.20.20). Daher gibt es noch genug IP-Adressen im oberen und unteren IP-Adressenbereich, die für eine statische Adressierung genutzt werden können.

Name	IP-Adressbereich
DMZ	192.168.20.1 – 255
LAN Segment 0	192.168.22.1 – 255
LAN Segment 1	192.168.23.1 – 255
LAN Segment 2	192.168.21.1 – 255

Tabelle 1: IP-Adressbereiche der Netzwerke

2.2.3. Clients

Alle Clients und der Server der Testumgebung haben Zugriff auf Webseiten im Internet. Zusätzlich haben alle Clients Zugriff auf den Webserver, sowie Zugriff auf den FTP-Server des Servers der DMZ. Alle Clients aus den LAN-Segmenten 0-2 können einen LAN-Segment übergreifenden Ping ausführen. Ein Ping aus der DMZ in eines der LAN-Segmente 0-2 ist nicht möglich. Der Web- und der FTP-Dienst des Servers der DMZ sollen aus dem Internet erreicht werden. Alle Clients aus den LAN-Segmenten 1-2 haben uneingeschränkten Zugriff auf das LAN-Segment 0.

2.2.4. Router

Die Router der Testumgebung dienen als Verbindung der verschieden Netzwerke aus Abbildung 1. Der Router „Firewall“ stellt die Schnittstelle zum Internet dar. Die Netzwerkkarte „eth1“ dieses Routers erhält eine IP-Adresse über einen DHCP-Server des externen Netzwerkes. Alle weiteren IP-Adressen der Netzwerkkarten von den Routern, sowie die DHCP- und DNS-Services sind in Tabelle 2 gelistet.

Router	Netzwerkkarte	IP-Adresse	DHCP-Server für	DNS-Server für
Firewall	eth0	DHCP aus Internet	192.168.20.0/24	192.168.20.0/24
	eth1	192.168.20.1		
R0	eth0	192.168.22.1	192.168.21.0/24 192.168.22.0/24	192.168.21.0/24 192.168.22.0/24
	eth1	192.168.21.1		
	eth2	192.168.20.2		
R1	eth0	192.168.23.1	192.168.23.0/24	192.168.23.0/24
	eth1	192.168.22.2		

Tabelle 2: Router der Testumgebung

In der Testumgebung sollen mehrere verschiedene Versionen von fli4l und auch verschiedene Kernel ausprobiert werden. Daher erhalten die Router verschiedene Versionen bzw. jeweils andere Kernel.

2.3. Aufbau der Infrastruktur

2.3.1. Auswahl des Client-Betriebssystems

Da die Testumgebung an alle Teilnehmer des Integrationsprojekts verteilt wird, wird ein Betriebssystem benötigt, welches kostenfrei verfügbar ist. Des Weiteren sollte das Betriebssystem mit möglichst wenigen Systemressourcen lauffähig sein. Die Studierenden der FHDW besitzen jeweils ein Notebook mit der Hardwarekonfiguration, welche der in Punkt 2.3.5, Notebook 1 beschrieben ist. Daher sollte die Testumgebung im eingeschalteten Zustand nicht mehr als 2 GB Arbeitsspeicher benötigen. Um die Router-Konfigurationen benutzerfreundlich testen zu können, benötigt das Betriebssystem eine Terminalfunktion und einen Webbrowser. Auf dem System in der DMZ werden ein Webserver und ein FTP-Server benötigt. Die Linux Distribution CentOS 6.3 bietet die oben genannten Anforderungen und wurde somit als Betriebssystem für die Clients und den Server ausgewählt.

2.3.2. Konfiguration der Router

Die Router der Testumgebung laufen als virtuelle Maschinen im VMware-Player. Das Betriebssystem inkl. deren Konfiguration wurde pro Router als CD-Image erstellt. Die wichtigste Konfigurationsdatei ist die „base.txt“, die für jede Router-Konfiguration benötigt wird. In dieser Datei wird die maßgebliche Konfiguration gespeichert. Weitere Konfigurationsdateien werden nur benötigt, wenn zusätzliche Pakete für spezielle Funktionalitäten genutzt werden. Weitere Funktionalitäten sind z.B. die Bereitstellung eines DHCP- und DNS-Servers oder das Fungieren eines Routers als DHCP-Client. Hierfür werden die Konfigurationsdateien „dns_dhcp.txt“ und „dhcp_client.txt“ genutzt.

Für die Konfiguration des Routers „Firewall“ werden Einstellungen in der Konfigurationsdatei „base.txt“ (siehe Tabelle 3) vorgenommen. Der Router „Firewall“ läuft mit der fli4l-Version 3.6.2.

Eintrag	Beschreibung
General Settings:	
HOSTNAME=‘firewall‘	Name des fli4l-Routers
PASSWORD=‘fli4l‘	ROOT-Passwort
BOOT_TYPE=‘cd‘	Die Art des Boot-Mediums muss hier auf „cd“ gestellt sein, da die Router-Konfiguration von einem CD-Image geladen werden soll.
MOUNT_BOOT=‘ro‘	Die Zugriffsart auf das Boot-Medium muss für ein CD-Image auf „ro“ für „read only“ gestellt werden, da von einem CD-Image nur gelesen werden kann.
KERNEL_VERSION=‘2.6.32.59‘	Für den Router „Firewall“ wird die Kernel-Version „2.6.32.59“ verwendet.
Ehernet card drivers:	
NET_DRV_N='2'	Anzahl der verfügbaren Netzwerkkarten-Treiber
NET_DRV_1='pcnet32'	1. Netzwerkkartentreiber (Standardtreiber)
NET_DRV_2='e1000'	2. Netzwerkkartentreiber (neuer VMware-fähiger Treiber)
Ether networks used with IP protocol:	
IP_NET_N='2'	Anzahl der verschiedenen IP-Netzwerke
IP_NET_1='192.168.20.1/24'	IP-Adresse der Netzwerkkarte des Routers im 1. Netzwerk
IP_NET_1_DEV='eth0'	Netzwerkkartenname für die Netzwerkkarte im 1. Netzwerk
IP_NET_2='dhcp'	Die IP-Adresse der Netzwerkkarte im 2. Netzwerk wird über DHCP bezogen.

IP_NET_2_DEV='eth1'	Netzwerkkartenname für die Netzwerkkarte im 2. Netzwerk
Additional routes:	
IP_ROUTE_N='1'	Anzahl der statischen Routen
IP_ROUTE_1='0.0.0.0/0 eth1'	Alle Pakete, die als Ziel ein nicht bekanntes Netzwerk haben, werden über die Netzwerkkarte „eth1“ weitergeleitet.
Packetfilter configuration:	
PF_NEW_CONFIG='yes'	Paketfilterregeln werden in der neuen Schreibweise verfasst.
PF_INPUT_POLICY='REJECT'	Alle eingehenden Pakete ohne vorhandene Eingangs-Regel, werden an den Sender zurückgewiesen.
PF_INPUT_ACCEPT_DEF='yes'	Default-Regeln des Routers werden verwendet.
PF_INPUT_LOG='no'	Das Protokollieren der eingehenden Pakete ist deaktiviert.
PF_INPUT_N='1'	Anzahl der Regeln für eingehende Pakete
PF_INPUT_1='IP_NET_1 ACCEPT'	1. eingehende Regel: Alle eingehenden Pakete aus dem IP_NET_1 werden akzeptiert.
PF_FORWARD_POLICY='REJECT'	Alle Pakete, für die es keine Weiterleitungs-Regel gibt, werden an den Sender zurückgewiesen.
PF_FORWARD_ACCEPT_DEF='yes'	Default-Regeln des Routers werden verwendet.
PF_FORWARD_LOG='no'	Das Protokollieren der Weiterleitung ist deaktiviert.
PF_FORWARD_N='4'	Anzahl der Weiterleitungs-Regeln
PF_FORWARD_1= 'any 192.168.20.20:80 ACCEPT'	1. Weiterleitungs-Regel: Jedes Paket welches über den Port 80 (http) an den Server adressiert ist, wird weitergeleitet.
PF_FORWARD_2= 'IP_NET_1 any:80 ACCEPT'	2. Weiterleitungs-Regel: Jedes Paket aus dem IP_NET_1, welches über den Port 80 gesendet wird, wird weitergeleitet.
PF_FORWARD_3='IP_NET_1 193.22.68.243:8080 ACCEPT'	3. Weiterleitungs-Regel: Jedes Paket aus dem IP_NET_1, welches an den Proxy-Server der FHDW gesendet wird, wird weitergeleitet.
PF_FORWARD_4= 'prot:icmp ACCEPT'	4. Weiterleitungs-Regel: Jedes Paket, welches ICMP-Inhalt (z.B. für Pings) besitzt, wird weitergeleitet.
PF_POSTROUTING_N='1'	Anzahl der Maskierungs-Regeln
PF_POSTROUTING_1= 'IP_NET_1 MASQUERADE'	Alle Pakete aus dem IP_NET_1 werden maskiert.
Port forwarding:	
PORTFW_N='2'	Anzahl der Port-Weiterleitungs-Regeln
PORTFW_1_TARGET='21'	1. Port-Weiterleitungsregel: Jedes Paket, welches über TCP-Port 21 (FTP) an den Router gesendet wird, wird an den FTP-Server des Servers in der DMZ weitergeleitet.
PORTFW_1_NEW_TARGET= '192.168.20.20:21'	
PORTFW_1_PROTOCOL='tcp'	
PORTFW_2_TARGET='80'	2. Port-Weiterleitungsregel: Jedes Paket, welches über TCP-Port 80 (http) an den Router gesendet wird, wird an den Server in der DMZ weitergeleitet.
PORTFW_2_NEW_TARGET= '192.168.20.20:80'	
PORTFW_2_PROTOCOL='tcp'	
Domain configuration	
DOMAIN_NAME='lan.firewall'	Domänenname des Routers
DNS_FORWARDERS=''	Kein statischer DNS-Server

Tabelle 3: Relevante Einstellungen der Konfigurationsdatei "base.txt" vom Router Firewall

Da der Router „Firewall“ den DHCP-Server und den DNS-Server für das Netzwerk „DMZ“ bereitstellen soll, müssen Einstellungen in der Konfigurationsdatei „dns_dhcp.txt“ (siehe Tabelle 4) vorgenommen werden.

Eintrag	Beschreibung
Hosts:	
HOSTS_N='1'	Anzahl statischer DHCP-Hosts
HOST_1_NAME='server'	Der Server bekommt eine eigene Zuordnung
HOST_1_IP4='192.168.20.20'	Und eine Statisch zugewiesene IP-Adresse, diese ist wichtig, da das Port-Forwarding sonst nicht funktioniert.
HOST_1_MAC= '00:0C:29:85:38:46'	Die Regel wird an die MAC-Adresse des Servers gebunden.
DNS configuration:	
OPT_DNS='yes'	Der DNS-Server des Routers wird eingeschaltet.
DNS_LISTENIP_N='1'	Anzahl an IP-Adressen über die der DNS-Server erreichbar ist.
DNS_LISTENIP_1='192.168.20.1'	1. IP-Adresse des DNS-Servers
DHCP-Server configuration:	
DHCP_RANGE_N='0'	Die Firewall vergibt keine dynamischen IP-Adressen.

Tabelle 4: Relevante Einstellungen der Konfigurationsdatei "dns_dhcp.txt" vom Router Firewall

Die Netzwerkkarte des Routers „Firewall“ die zum Internet verbindet, erhält ihre IP-Adresse von einem DHCP-Server aus dem Internet, daher muss der Router auch als DHCP-Client fungieren. Diese Einstellungen müssen in der Konfigurationsdatei „dhcp_client.txt“ vorgenommen werden.

Eintrag	Beschreibung
DHCP client:	
OPT_DHCP_CLIENT='yes'	DHCP-Client wird aktiviert
DHCP_CLIENT_TYPE='dhcpcd'	Der DHCP-Typ
DHCP_CLIENT_N='1'	Einer NIC wird die IP-Adresse per DHCP zugewiesen.
DHCP_CLIENT_1_IF='IP_NET_2_DEV'	Festlegung, welcher NIC die Adresse zugewiesen wird.
DHCP_CLIENT_1_ROUTE='default'	Das Default-Gateway ist die IP-Adresse des DHCP-Servers
DHCP_CLIENT_1_USEPEERDNS='yes'	Der DNS-Server des Internets wird genutzt

Tabelle 5: Relevante Einstellungen der Konfigurationsdatei "dhcp_client.txt" vom Router Firewall

Für die Konfiguration des Routers „R0“ werden Einstellungen in der Konfigurationsdatei „base.txt“ (siehe Tabelle 6) vorgenommen. Der Router „R0“ läuft mit der Tarball-Version 3.9.0.

Eintrag	Beschreibung
General Settings:	
HOSTNAME=‘r0‘	Name des fli4l-Routers
PASSWORD=‘fli4l‘	ROOT-Passwort
BOOT_TYPE=‘cd‘	Die Art des Boot-Mediums muss hier auf „cd“ gestellt sein, da die Router-Konfiguration von einem CD-Image geladen werden soll.
MOUNT_BOOT=‘ro‘	Die Zugriffsart auf das Boot-Medium muss für ein CD-Image auf „ro“ für „read only“ gestellt werden, da von einem CD-Image nur gelesen werden kann.

KERNEL_VERSION=‘ 3.7.4 ‘	Für den Router „R0“ wird die Kernel-Version „3.7.4“ verwendet (Aus der Tarball Version).
Ehernet card drivers:	
NET_DRV_N='2'	Anzahl der verfügbaren Netzwerkkarten-Treiber
NET_DRV_1='pcnet32'	1. Netzwerkkartentreiber (Standardtreiber)
NET_DRV_2='e1000'	2. Netzwerkkartentreiber (neuer VMware-fähiger Treiber)
Ether networks used with IP protocol:	
IP_NET_N='3'	Anzahl der verschiedenen IP-Netzwerke
IP_NET_1='192.168.22.1/24'	IP-Adresse der Netzwerkkarte des Routers im 1. Netzwerk
IP_NET_1_DEV='eth0'	Netzwerkkartenname für die Netzwerkkarte im 1. Netzwerk
IP_NET_2='192.168.21.1/24'	Die IP-Adresse der Netzwerkkarte im 2. Netzwerk
IP_NET_2_DEV='eth1'	Netzwerkkartenname für die Netzwerkkarte im 2. Netzwerk
IP_NET_3='192.168.20.2/24'	Die IP-Adresse der Netzwerkkarte im 3. Netzwerk
IP_NET_3_DEV='eth2'	Netzwerkkartenname für die Netzwerkkarte im 3. Netzwerk
Additional routes:	
IP_ROUTE_N='4'	Anzahl der statischen Routen
IP_ROUTE_1='192.168.22.0/24 eth0'	Anfragen ans S0 werden an eth0 weitergeleitet.
IP_ROUTE_2='192.168.20.0/24 eth2'	Anfragen an die DMZ werden auch an eth2 weitergeleitet.
IP_ROUTE_3='192.168.21.0/24 eth1'	Anfragen ans S2 werden an eth1 weitergeleitet.
IP_ROUTE_4='0.0.0.0/0 192.168.20.1'	Alle Pakete, die als Ziel ein nicht bekanntes Netzwerk haben, werden an die „Firewall“ weitergeleitet.
Packetfilter configuration:	
PF_NEW_CONFIG='yes'	Paketfilterregeln werden in der neuen Schreibweise verfasst.
PF_INPUT_POLICY='REJECT'	Alle eingehenden Pakete ohne vorhandene Eingangs-Regel, werden an den Sender zurückgewiesen.
PF_INPUT_ACCEPT_DEF='yes'	Default-Regeln des Routers werden verwendet.
PF_INPUT_LOG='no'	Das Protokollieren der eingehenden Pakete ist deaktiviert.
PF_INPUT_N='2'	Anzahl der Regeln für eingehende Pakete
PF_INPUT_1='IP_NET_1 ACCEPT'	1. eingehende Regel: Alle eingehenden Pakete aus dem IP_NET_1 werden akzeptiert.
PF_INPUT_2='IP_NET_2 ACCEPT'	2. eingehende Regel: Alle eingehenden Pakete aus dem IP_NET_2 werden akzeptiert.
PF_FORWARD_POLICY='REJECT'	Alle Pakete, für die es keine Weiterleitungs-Regel gibt, werden an den Sender zurückgewiesen.
PF_FORWARD_ACCEPT_DEF='yes'	Default-Regeln des Routers werden verwendet.
PF_FORWARD_LOG='no'	Das Protokollieren der Weiterleitung ist deaktiviert.
PF_FORWARD_N='8'	Anzahl der Weiterleitungs-Regeln
PF_FORWARD_1='IP_NET_2 192.168.20.20:21 ACCEPT'	1. Weiterleitungs-Regel: Jedes Paket welches über den Port 21 (ftp) an den Server adressiert ist, wird weitergeleitet.
PF_FORWARD_2='IP_NET_2 any:80 ACCEPT'	2. Weiterleitungs-Regel: Jedes Paket aus dem IP_NET_2, welches über den Port 80 gesendet wird, wird weitergeleitet.

PF_FORWARD_3='IP_NET_2 193.22.68.243:8080 ACCEPT'	3. Jedes Paket aus dem IP_NET_2, welches an den Proxy-Server der FHDW gesendet wird, wird weitergeleitet.
PF_FORWARD_4='IP_NET_2 192.168.22.0/24 ACCEPT'	4. Weiterleitungs-Regel: Jedes Paket aus dem IP_NET_2, welches an das S0 adressiert ist, wird weitergeleitet.
PF_FORWARD_5='IP_NET_1 192.168.20.20:80 ACCEPT'	5. Weiterleitungs-Regel: Jedes Paket welches über den Port 80 (http) an den Server adressiert ist, wird weitergeleitet.
PF_FORWARD_6='IP_NET_1 any:80 ACCEPT'	6. Weiterleitungs-Regel: Jedes Paket aus dem IP_NET_1, welches über den Port 80 gesendet wird, wird weitergeleitet.
PF_FORWARD_7='IP_NET_1 193.22.68.243:8080 ACCEPT'	7. Jedes Paket aus dem IP_NET_1, welches an den Proxy-Server der FHDW gesendet wird, wird weitergeleitet.
PF_FORWARD_8='prot:icmp ACCEPT'	8. Weiterleitungs-Regel: Jedes Paket, welches ICMP-Inhalt (z.B. für Pings) besitzt, wird weitergeleitet.
PF_POSTROUTING_N='2'	Anzahl der Maskierungs-Regeln
PF_POSTROUTING_1= 'IP_NET_1 MASQUERADE'	Alle Pakete aus dem IP_NET_1 werden maskiert.
PF_POSTROUTING_2= 'IP_NET_2 MASQUERADE'	Alle Pakete aus dem IP_NET_2 werden maskiert.
Domain configuration	
DNS_FORWARDERS='192.168.20.1'	Ein statisch eingetragener DNS-Server

Tabelle 6: Relevante Einstellungen der Konfigurationsdatei "base.txt" vom Router R0

Da der Router „R0“ den DHCP-Server und den DNS-Server für das Netzwerk „S0“ und „S2“ bereitstellen soll, müssen Einstellungen in der Konfigurationsdatei „dns_dhcp.txt“ (siehe Tabelle 7) vorgenommen werden.

Eintrag	Beschreibung
Hosts:	
HOSTS_N='0'	Anzahl statischer DHCP-Hosts
DNS configuration:	
OPT_DNS='yes'	Der DNS-Server wird eingeschaltet.
DNS_LISTENIP_N='2'	Anzahl an IP-Adressen über die der DNS-Server erreichbar ist.
DNS_LISTENIP_1='192.168.22.1'	1. IP-Adresse des DNS-Servers
DNS_LISTENIP_2='192.168.21.1'	2. IP-Adresse des DNS-Servers
DHCP-Server configuration:	
DHCP_RANGE_N='2'	Anzahl an dynamisch vergebenen IP-Adressbereichen.
DHCP_RANGE_1_NET='IP_NET_1'	Netzwerk, an dem die Adressvergabe stattfinden soll.
DHCP_RANGE_1_START= '192.168.22.100'	Anfang des dynamisch zu vergebenden Adressbereichs.
DHCP_RANGE_1_END= '192.168.22.150'	Ende des dynamisch zu vergebenden Adressbereichs.
DHCP_RANGE_1_DNS_SERVER1= '192.168.22.1'	Primärer DNS-Server für den Adressbereich
DHCP_RANGE_1_DNS_SERVER2=''	Sekundärer DNS-Server für den Adressbereich
DHCP_RANGE_1_GATEWAY= '192.168.22.1'	Gateway für den Adressbereich

DHCP_RANGE_2_NET='IP_NET_2'	Netzwerk, an dem die Adressvergabe stattfinden soll.
DHCP_RANGE_2_START= '192.168.21.100'	Anfang des dynamisch zu vergebenden Adressbereichs.
DHCP_RANGE_2_END= '192.168.21.150'	Ende des dynamisch zu vergebenden Adressbereichs.
DHCP_RANGE_2_DNS_SERVER1= '192.168.21.1'	Anzahl an dynamisch vergebenen IP-Adressbereichen.
DHCP_RANGE_2_DNS_SERVER2=''	Netzwerk, an dem die Adressvergabe stattfinden soll.
DHCP_RANGE_2_GATEWAY= '192.168.21.1'	Anfang des dynamisch zu vergebenden Adressbereichs.

Tabelle 7: Relevante Einstellungen der Konfigurationsdatei "dns_dhcp.txt" vom Router R0

Für die Konfiguration des Routers „R1“ werden Einstellungen in der Konfigurationsdatei „base.txt“ (siehe Tabelle 8) vorgenommen. Der Router „R1“ läuft ebenfalls mit der fli4l-Version 3.6.2.

Eintrag	Beschreibung
General Settings:	
HOSTNAME=‘r1‘	Name des fli4l-Routers
PASSWORD=‘fli4l‘	ROOT-Passwort
BOOT_TYPE=‘cd‘	Die Art des Boot-Mediums muss hier auf „cd“ gestellt sein, da die Router-Konfiguration von einem CD-Image geladen werden soll.
MOUNT_BOOT=‘ro‘	Die Zugriffsart auf das Boot-Medium muss für ein CD-Image auf „ro“ für „read only“ gestellt werden, da von einem CD-Image nur gelesen werden kann.
KERNEL_VERSION=‘2.6.16.62‘	Für den Router „R1“ wird die Kernel-Version „2.6.16.62“ verwendet (Alter Kernel).
Ehernet card drivers:	
NET_DRV_N='2'	Anzahl der verfügbaren Netzwerkkarten-Treiber
NET_DRV_1='pcnet32'	1. Netzwerkkartentreiber (Standardtreiber)
NET_DRV_2='e1000'	2. Netzwerkkartentreiber (neuer VMware-fähiger Treiber)
Ether networks used with IP protocol:	
IP_NET_N='2'	Anzahl der verschiedenen IP-Netzwerke
IP_NET_1='192.168.23.1/24'	IP-Adresse der Netzwerkkarte des Routers im 1. Netzwerk
IP_NET_1_DEV='eth0'	Netzwerkkartenname für die Netzwerkkarte im 1. Netzwerk
IP_NET_2='192.168.22.2/24'	Die IP-Adresse der Netzwerkkarte im 2. Netzwerk wird über DHCP bezogen.
IP_NET_2_DEV='eth1'	Netzwerkkartenname für die Netzwerkkarte im 2. Netzwerk
Additional routes:	
IP_ROUTE_N='3'	Anzahl der statischen Routen
IP_ROUTE_1='192.168.23.0/24 eth0'	Anfragen ans S1 werden an eth0 weitergeleitet.
IP_ROUTE_2='192.168.22.0/24 eth1'	Anfragen ans S0 werden an eth1 weitergeleitet.
IP_ROUTE_3='0.0.0.0/0	Alle Pakete, die als Ziel ein nicht bekanntes Netzwerk haben,

192.168.22.1'	werden an „R0“ weitergeleitet.
Packetfilter configuration:	
PF_NEW_CONFIG='yes'	Paketfilterregeln werden in der neuen Schreibweise verfasst.
PF_INPUT_POLICY='REJECT'	Alle eingehenden Pakete ohne vorhandene Eingangs-Regel, werden an den Sender zurückgewiesen.
PF_INPUT_ACCEPT_DEF='yes'	Default-Regeln des Routers werden verwendet.
PF_INPUT_LOG='no'	Das Protokollieren der eingehenden Pakete ist deaktiviert.
PF_INPUT_N='1'	Anzahl der Regeln für eingehende Pakete
PF_INPUT_1='IP_NET_1 ACCEPT'	1. eingehende Regel: Alle eingehenden Pakete aus dem IP_NET_1 werden akzeptiert.
PF_FORWARD_POLICY='REJECT'	Alle Pakete, für die es keine Weiterleitungs-Regel gibt, werden an den Sender zurückgewiesen.
PF_FORWARD_ACCEPT_DEF='yes'	Default-Regeln des Routers werden verwendet.
PF_FORWARD_LOG='no'	Das Protokollieren der Weiterleitung ist deaktiviert.
PF_FORWARD_N='5'	Anzahl der Weiterleitungs-Regeln
PF_FORWARD_1='IP_NET_1 192.168.20.20:21 ACCEPT'	1. Weiterleitungs-Regel: Jedes Paket welches über den Port 21 (ftp) an den Server adressiert ist, wird weitergeleitet.
PF_FORWARD_2='IP_NET_1 any:80 ACCEPT'	2. Weiterleitungs-Regel: Jedes Paket aus dem IP_NET_1, welches über den Port 80 gesendet wird, wird weitergeleitet.
PF_FORWARD_3='IP_NET_1 193.22.68.243:8080 ACCEPT'	3. Jedes Paket aus dem IP_NET_1, welches an den Proxy-Server der FHDW gesendet wird, wird weitergeleitet.
PF_FORWARD_4='IP_NET_1 192.168.22.0/24 ACCEPT'	4. Weiterleitungs-Regel: Jedes Paket aus dem IP_NET_1, welches an das S0 adressiert ist, wird weitergeleitet.
PF_FORWARD_5='prot:icmp ACCEPT'	5. Weiterleitungs-Regel: Jedes Paket, welches ICMP-Inhalt (z.B. für Pings) besitzt, wird weitergeleitet.
PF_POSTROUTING_N='1'	Anzahl der Maskierungs-Regeln
PF_POSTROUTING_1='IP_NET_1 MASQUERADE'	Alle Pakete aus dem IP_NET_1 werden maskiert.
Domain configuration	
DNS_FORWARDERS='192.168.22.1'	Ein statisch eingetragener DNS-Server

Tabelle 8: Relevante Einstellungen der Konfigurationsdatei "base.txt" vom Router R1

Da der Router „R1“ den DHCP-Server und den DNS-Server für das Netzwerk „S1“ bereitstellen soll, müssen Einstellungen in der Konfigurationsdatei „dns_dhcp.txt“ (siehe Tabelle 9) vorgenommen werden.

Eintrag	Beschreibung
Hosts:	
HOSTS_N='0'	Anzahl statischer DHCP-Hosts
DNS configuration:	
OPT_DNS='yes'	Der DNS-Server wird eingeschaltet.
DNS_LISTENIP_N='2'	Anzahl an IP-Adressen über die der DNS-Server erreichbar ist.
DNS_LISTENIP_1='192.168.23.1'	1. IP-Adresse des DNS-Servers
DHCP-Server configuration:	

DHCP_RANGE_N='1'	Anzahl an dynamisch vergebenen IP-Adressbereichen.
DHCP_RANGE_1_NET='IP_NET_1'	Netzwerk, an dem die Adressvergabe stattfinden soll.
DHCP_RANGE_1_START= '192.168.23.100'	Anfang des dynamisch zu vergebenden Adressbereichs.
DHCP_RANGE_1_END= '192.168.23.150'	Ende des dynamisch zu vergebenden Adressbereichs.
DHCP_RANGE_1_DNS_SERVER1= '192.168.23.1'	Primärer DNS-Server für den Adressbereich
DHCP_RANGE_1_DNS_SERVER2=''	Sekundärer DNS-Server für den Adressbereich
DHCP_RANGE_1_GATEWAY= '192.168.23.1'	Gateway für den Adressbereich

Tabelle 9: Relevante Einstellungen der Konfigurationsdatei "dns_dhcp.txt" vom Router R1

2.3.3. Konfiguration der Clients

Die Client-Rechner (in Abbildung 1 in Blau eingezeichnet) wurden alle mit CentOS 6.3 ausgestattet. Während bei Client S0 - S1 nur eine Standard-Installation vollzogen wurde, wurden auf dem Server zusätzlich und nachträglich noch ein HTTP-Daemon (httpd) und ein FTP-Server (vsftp) installiert. Um die DNS-Auflösung und die Verbindung ins Internet testen zu können, wurden in allen Clients die Proxy-Einstellungen von Firefox, den Voraussetzungen des FHDW-Netzwerkes entsprechend, vorgenommen. Dabei wurde außer den Loopback-Adressen noch der Server als Ausnahme eingetragen.

- Proxy Einstellungen: 193.22.68.243:8080 für alle Protokolle
- Proxy umgehen für: 127.0.0.1, localhost, 192.168.20.20

Auf allen Clients kann man sich unter folgender Benutzeranmeldung einloggen:

- Username: „User1“
- Passwort: „user1“

Werden root-Rechte benötigt, so erhält man diese mit dem Benutzer „root“ und dem Passwort „qwertz“.

2.3.4. Erstellung der virtuellen Maschinen

Clients

Für die Clients wurde das CentOS-Image (Version 6.3, Stand Jan. 2013) zur Installation verwendet. Die VMs wurden mit 10 GB Festplatte und jeweils 256 MB RAM ausgestattet. Die Netzwerkkarten wurden an die, der Abbildung 1 entsprechenden LAN-Segmente angeschlossen. Als Bootmedium wurde das von http://www.centos.org heruntergeladene CD-ROM-Abbild CentOS-6.3-i386-LiveCD.iso eingebunden.

Router

Um sich einen eigenes fli4l-Build zu erstellen lädt man sich zunächst die gewünschten Pakete von der fli4l-Projektseite herunter.[5] Hier sind das „Base“-Paket, sowie ein Kernel zwingend notwendig. Alle

[5] net(t)work(s) e.V., „fli4l,“ 2013. [Online]. Available: http://www.fli4l.de/download/stabile-version/pakete/.

weiteren Pakete sind optional. Anschließend öffnet man die gewünschte Konfigurationsdatei (z.B. die „config/base.txt“) und bearbeitet diese entsprechend.

Bei der „config/base.txt“ ist es u.a. wichtig die Kernel-Version anzugeben, die man heruntergeladen hat und ebenso der gewünschte Boot-Typ (in unserem Fall standardmäßig: „cd“). Wenn man sich für den Boot-Typ „cd“ entschieden hat muss man noch beachten, unter dem Punkt „MOUNT_BOOT“ die Option „ro“ für „read-only“ auszuwählen. Nach Abschluss der Konfiguration, wird die Datei „mkfli4l.bat“ geöffnet, um ein entsprechendes fli4l-Image zu erstellen. Hier gibt man den Pfad zum „config“- sowie „build“-Ordner an und kann noch weitere Parameter einstellen (siehe Abbildung 2).

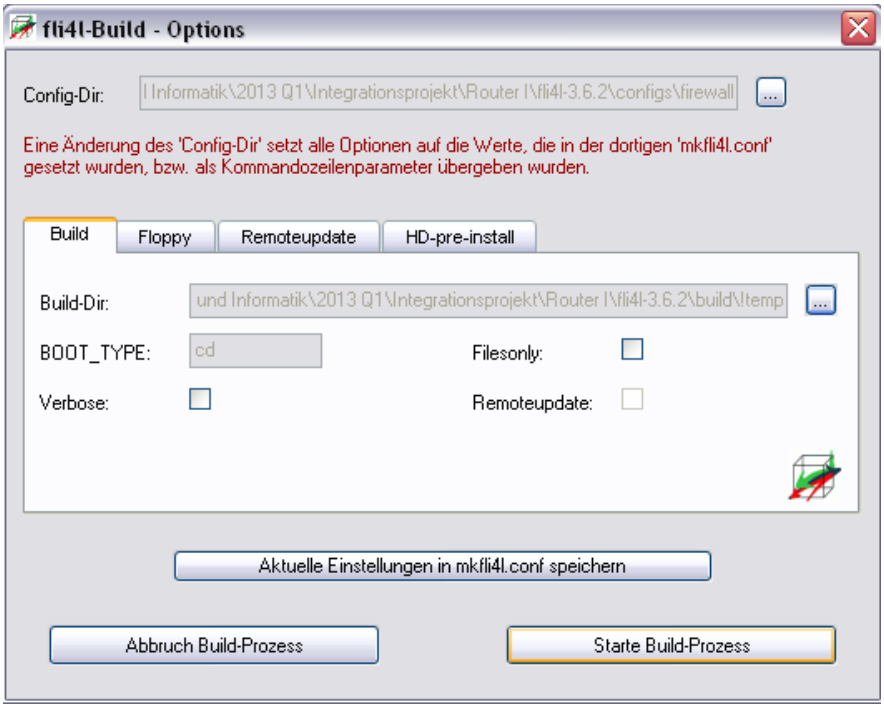


Abbildung 2: Programm zum Erstellen von fli4l-Builds

Danach klickt man unten rechts auf "Starte Build-Prozess", um selbigen zu starten.

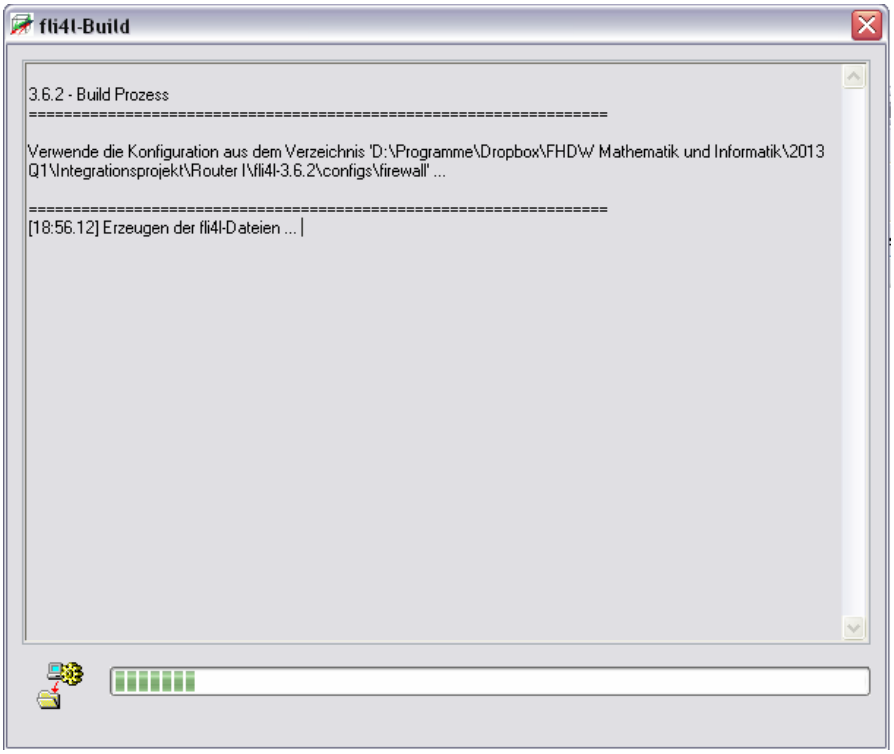

fli4l-Build
3.6.2 - Build Prozess
Verwende die Konfiguration aus dem Verzeichnis 'D:\Programme\Dropbox\FHDW Mathematik und Informatik\2013 Q1\Integrationsprojekt\Router I\fli4l-3.6.2\configs\firewall' ...
[18:56.12] Erzeugen der fli4l-Dateien ...

Abbildung 3: Programm während der Erstellung eines fli4l-Builds

Wenn der Build-Prozess erfolgreich war, wird im zuvor ausgewählten Build-Ordner eine Image-Datei namens 'fli4l.iso' erstellt, die man in eine bestehende virtuelle Maschine einfügen kann. Im Rahmen des Projekts wurden mit Hilfe solcher 'fli4l.iso'-Dateien neue virtuelle Maschinen, die als Router fungieren, mit dem VMware Player erstellt. Die VMs wurden ohne Festplatte und jeweils 64MB RAM ausgestattet. Die Netzwerkkarten wurden an die, der Abbildung 1 entsprechenden LAN-Segmente angeschlossen, mit Ausnahme der Netzwerkkarte zum realen Netzwerk. Für diese wurde eine Netzwerk-Brücke eingerichtet.

2.3.5. Testfälle

Die Infrastruktur wurde auf verschiedenen Notebooks erstellt, auf denen die virtuellen Maschinen installiert und konfiguriert worden sind. Die Testfälle wurden auf diese drei Notebooks ausgeführt:

Notebook 1:

- Intel Core i5 M520 2,4 GHz
- 4GB DDR3-RAM
- Intel OnBoard Grafik mit Shared Memory
- 250GB Festplatte
- Vorinstalliertes Windows XP SP3 Professional-Betriebssystem für Studierende der FHDW

Notebook 2:

- Intel Core i5 M 520 2,4 GHz
- 6GB DDR3-RAM
- Intel OnBoard Grafik mit Shared Memory

- 250GB Festplatte
- Selbstinstalliertes und -konfiguriertes Windows 8 Professional Betriebssystem

Notebook 3:

- Intel Core i7-2640M 2,8 GHz
- 8GB DDR3-RAM
- Intel OnBoard Grafik mit Shared Memory
- 500GB Festplatte
- Selbstinstalliertes und -konfiguriertes Windows 7 SP1 Professional Betriebssystem

In folgenden Tabellen werden die Testfälle für die Infrastruktur abgebildet. Tabelle 10 zeigt zum besseren Verständnis, Erläuterungen zu den Bezeichnungen der Tabellen 4-7.

Bezeichnung	IP-Adresse
S0	192.168.22.XXX (DHCP)
S1	192.168.23.XXX (DHCP)
S2	192.168.21.XXX (DHCP)
R0	192.168.21.1 (eth1), 192.168.22.1 (eth0), 192.168.20.2 (eth2)
R1	192.168.22.2 (eth1), 192.168.23.1 (eth0)
Firewall	192.168.20.1 (eth1), 172.17.XXX.XXX (eth0,DHCP)
Server	192.168.20.20
Extern	Beliebiger Rechner im selben Netz wie die Firewall an eth0

Tabelle 10: Legende für die Testfälle

Um die LAN-Segment übergreifenden Pings zu gewährleisten, dienen die Testfälle aus der Tabelle 11.

Testfall	Erwartung
S1 → S0	kommt an
S1 → S2	kommt an
S1 → Server	kommt an
S1 → Internet	kommt an
S0 → S1	kommt an
S0 → S2	kommt an
S0 → Server	kommt an
S0 → Internet	kommt an
S2 → S0	kommt an
S2 → S1	kommt an
S2 → Server	kommt an
S2 → Internet	kommt an
Server → Internet	kommt an
Server → S0	kommt nicht an
Server → R0	kommt an
Server → S2	kommt nicht an

Tabelle 11: ICMP-Testfälle

Um den Zugriff auf die Webseiten des Servers und des Internets zu gewährleisten, wurden die Testfälle der Tabelle 12 durchgeführt.

Testfall	Erwartung
S1 → Internet (über proxy2:8080)	200 OK
S1 → Server (Server:80)	200 OK
S0 → Internet (über proxy2:8080)	200 OK
S0 → Server (Server:80)	200 OK
S2 → Internet (über proxy2:8080)	200 OK
S2 → Server (Server:80)	200 OK
Server → Internet (über proxy2:8080)	200 OK
Extern → Server (Server:80)	200 OK

Tabelle 12: TCP-Testfälle für http

Ein FTP-Zugriff aus allen LAN-Segmenten auf den Server sollte ermöglicht werden. Die Testfälle aus Tabelle 13 dienen zur Überprüfung des FTP-Zugriffes.

Testfall	Erwartung
S1 → Server (Server:21)	Verbindungsaufbau
S0 → Server (Server:21)	Verbindungsaufbau
S2 → Server (Server:21)	Verbindungsaufbau
Server → Internet (über proxy2:8080)	kein Verbindungsaufbau (proxy2 blockiert ftp)
Extern → Server (Server:21)	Verbindungsaufbau

Tabelle 13: TCP-Testfälle für ftp

Mit folgenden Testfällen wird die DNS-Auflösung geprüft.

Testfall	Erwartung
S1 → google.com	kommt an
S0 → google.com	kommt an
S2 → google.com	kommt an
R0 → google.com	kommt an
R1 → google.com	kommt an
Firewall → google.com	kommt an
Server → google.com	kommt an

Tabelle 14: DNS-Testfälle für Ping

2.4. Handbuch

2.4.1. Installation von VMware-Player

Der „VMware-Player“ ist ein Programm, um so genannte virtuelle Maschinen auf einem nativen System laufen zu lassen. So ist es bspw. möglich, auf einem bestehenden installierten und lauffähigen Betriebssystem, weitere Betriebssysteme in einer virtuellen Umgebung zu installieren. Für dieses Projekt nutzen wir dieses Programm, um die fli4l-Router, sowie weitere Hosts auf den Notebooks abzubilden.

Zunächst wird der "VMware Player" von der Hersteller-Internetseite heruntergeladen.[6] Dieser ist der derzeit in der Version 5.0.1 Build 894247 verfügbar. (Stand: 25.1.2013)

Anschließend startet man die heruntergeladene „VMware-player-5.0.1-894247.exe“ (in diesem Fall für Windows-Betriebssysteme) und führt das Setup aus.

Hier wählt man den gewünschten Installationspfad aus und kann noch diverse Einstellungen vornehmen, die jedoch nicht von besonderer Relevanz sind und installiert den "VMware Player".

2.4.2. Installation von ImDisk

Mit dem Programm „ImDisk Virtual Disk Driver“ lässt sich ein virtuelles Diskettenlaufwerk auf einem Windows-System installieren. Für dieses Projekt kann es genutzt werden, um bootfähige Disketten-Images von fli4l zu erstellen. Dies stellt eine Alternative zu den bestehenden CD-Images dar.

Zunächst wird der „ImDisk Virtual Disk Driver“ von der Hersteller-Internetseite heruntergeladen.[7] Dieser ist der derzeit in der Version 1.6.0 Build 2 verfügbar. (Stand: 25.1.2013)

Anschließend startet man die heruntergeladene „imdiskinst.exe“ und führt das Setup aus. Die Installation gestaltet sich recht einfach. Man klickt im erscheinenden Fenster auf „Ja“ und wartet bis der Installationsvorgang abgeschlossen ist. Nach erfolgreicher Installation erscheint ein Fenster, welches man mit „Ok“ bestätigt.

In der Windows-Systemsteuerung befindet sich nun ein weiterer Punkt namens „ImDisk Virtual Disk Driver“. Dort kann man nun virtuelle Diskettenlaufwerke anlegen, indem nach dem Starten unten links den Punkt „Mount new...“ auswählt. Im neuen Fenster (siehe Abbildung 4) wählt man die gewünschten Parameter aus und bestätigt mit „Ok“.

[6] VMware Inc., „VMware Downloads,“ 2013. [Online]. Available: http://www.vmware.com/go/downloadplayer/.
[7] LTR Data, „ImDisk Virtual Disk Driver Download,“ [Online]. Available: http://www.ltr-data.se/opencode.html/#ImDisk.

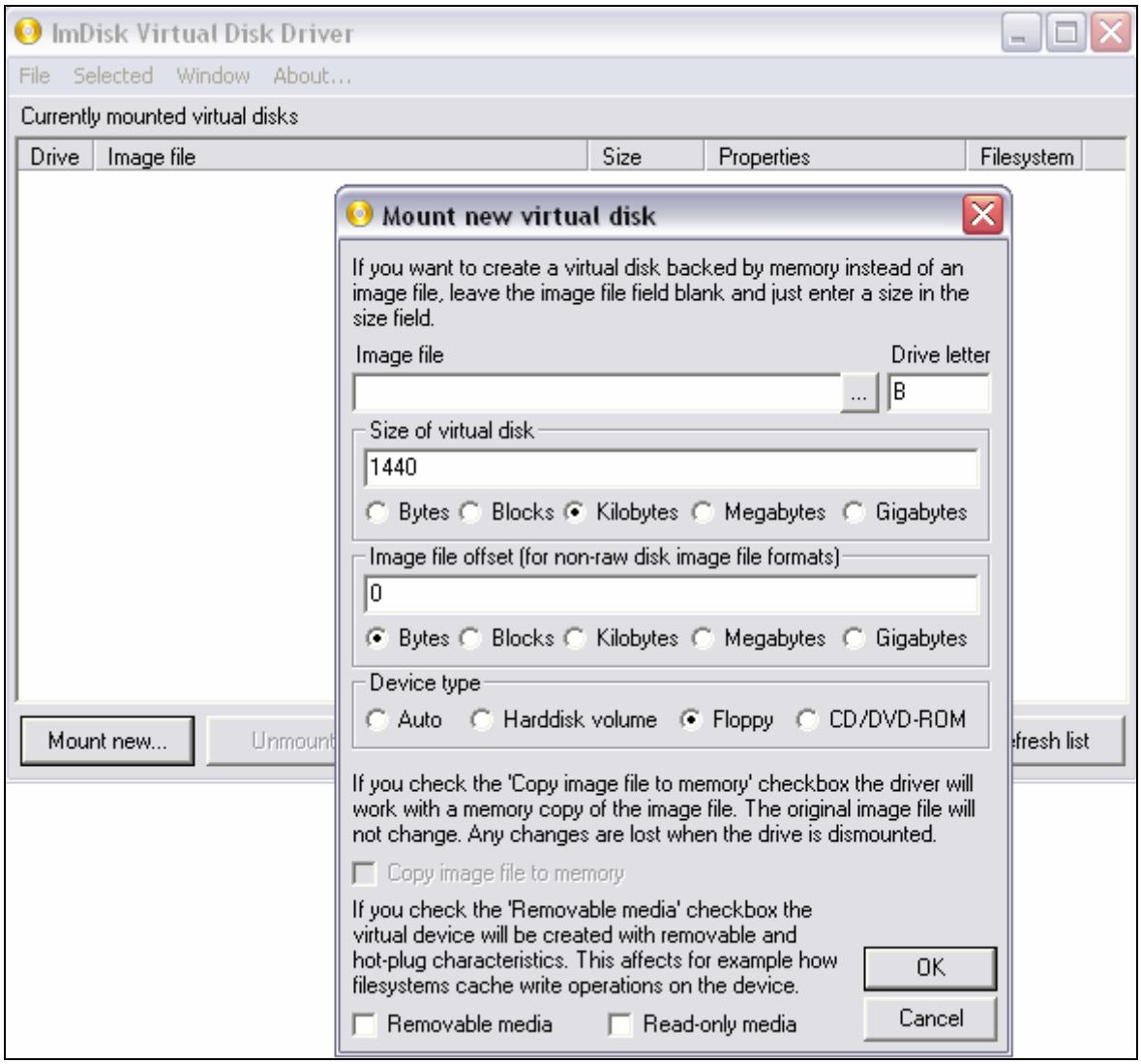

ImDisk Virtual Disk Driver
File Selected Window About...
Currently mounted virtual disks
Drive Image file Size Properties Filesystem
Mount new virtual disk
If you want to create a virtual disk backed by memory instead of an image file, leave the image file field blank and just enter a size in the size field.
Image file
Drive letter
B
Size of virtual disk
1440
Bytes Blocks Kilobytes Megabytes Gigabytes
Image file offset (for non-raw disk image file formats)
0
Bytes Blocks Kilobytes Megabytes Gigabytes
Device type
Auto Harddisk volume Floppy CD/DVD-ROM
If you check the 'Copy image file to memory' checkbox the driver will work with a memory copy of the image file. The original image file will not change. Any changes are lost when the drive is dismounted.
Copy image file to memory
If you check the 'Removable media' checkbox the virtual device will be created with removable and hot-plug characteristics. This affects for example how filesystems cache write operations on the device.
OK
Cancel
Removable media
Read-only media
Mount new...
Unmount
fresh list

Abbildung 4: Einhängen eines neuen virtuellen Diskettenlaufwerkes

Anschließend kann man bei fli4l eine Diskettenkonfiguration erstellen und kopiert diese in das virtuelle Diskettenlaufwerk. Bei „ImDisk Virtual Disk Driver“ kann man nun im Hauptfenster unten rechts auf „Save Image“ klicken, um ein Disketten-Image zu erstellen (*.img). Im neuen Fenster wählt man den unteren Punkt „Save disk image at offset 0 in new image file. (...)“ aus (siehe Abbildung 5). Um das Disketten-Image beim VMware Player einfügen muss man schließlich noch die Dateiendung von *.img auf *.flp ändern.

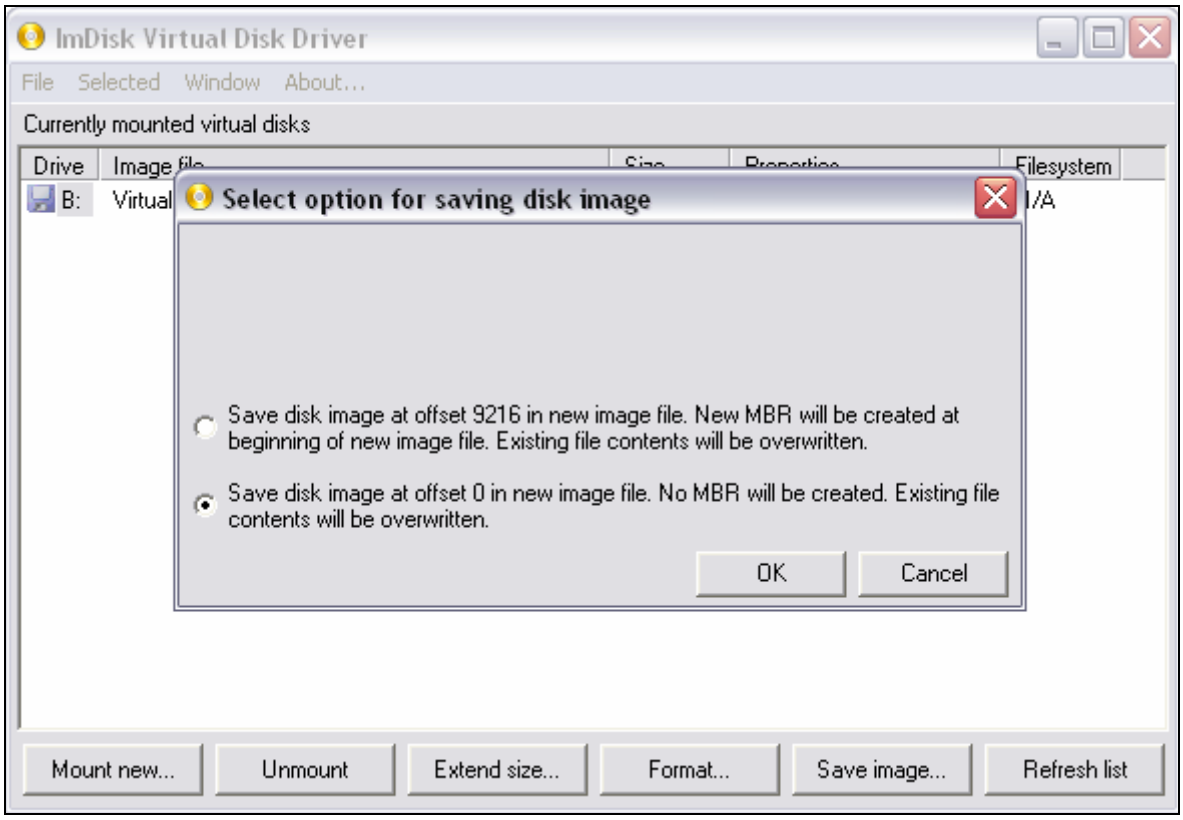

ImDisk Virtual Disk Driver
File Selected Window About...
Currently mounted virtual disks
Drive
Image file
Size
Properties
Filesystem
B:
Virtual
N/A
Select option for saving disk image
Save disk image at offset 9216 in new image file. New MBR will be created at beginning of new image file. Existing file contents will be overwritten.
Save disk image at offset 0 in new image file. No MBR will be created. Existing file contents will be overwritten.
OK
Cancel
Mount new...
Unmount
Extend size...
Format...
Save image...
Refresh list

Abbildung 5: Einstellung für ein Disketten-Image

2.4.3. Entpacken der virtuellen Maschinen

Die virtuellen Maschinen (VMs) der Testumgebung befinden sich in der Datei „VMs.7z“. Da die virtuellen Maschinen für die Testumgebung mit VMware realisiert sind, sind die VMs standardmäßig im Ordner „%USERPROFILE%\Documents\Virtual Machines“ zu finden. Die Informationen von jeder VM sind in jeweils einem eigenen Unterordner zu finden. Die Ordner der VMs können, so wie sie sind, in das Standardverzeichnis von VMware-Player kopiert werden. Eingebundene Images für Disketten oder CD-ROMs befinden sich in den Ordnern der jeweils zugehörigen VMs. Die VMs können gestartet werden, indem die .vmx-Datei ausgeführt wird (vorher aber sollten die LAN Segmente konfiguriert werden. Siehe Punkt 2.4.4.4).

2.4.4. Konfiguration der LAN-Segmente

Die LAN-Segmente werden bei VMware unter „%APPDATA%\Roaming\VMware\preferences.ini“ gespeichert. Um die Segmente auch auf dem Zielsystem einzurichten, muss zuerst die Zeile mit der Anzahl der Segmente kopiert oder im Zielsystem angepasst werden. VMware sollte erst danach gestartet werden, damit die Konfiguration übernommen wird. In diesem Projekt werden vier LAN Segmente verwendet.

Beispiel 1: pref.namedPVNs.count = "4"

Als nächstes müssen die jeweils zwei Zeilen der Segmente eingefügt werden. Die erste Zeile bezeichnet den Namen, die zweite eine ID, die auch in den VMs in der Adapterkonfiguration eingetragen ist. Über diese findet nachher die Zuordnung statt.

Beispiel 2: pref.namedPVNs0.name = "LAN Segment 1"
pref.namedPVNs0.pvnID = "52 8b c9 a0 9a 90 ed 18-09 ce 76 cb b7 0d 85 d1"

Danach kann VMware auf dem Zielsystem gestartet und die VMs geladen werden. Sie sind dann automatisch mit den LAN-Segmenten verbunden.

2.4.5. Starten der virtuellen Maschinen

VMware fragt nach dem Kopieren, ob die VMs kopiert oder verschoben worden sind. Wurden sie verschoben, werden netzwerkrelevante Eigenschaften der VMs (MAC-Adressen) beibehalten. Sollten sie kopiert worden sein, werden neue MAC-Adressen vergeben. Dadurch wird verhindert, dass identische MAC-Adressen mehrfach im Netzwerk vorkommen. Für unsere Zwecke sollten bei allen VMs „Verschoben“ ausgewählt werden. Eine Ausnahme bildet der Router „Firewall“, da dieser auch nach außen kommuniziert. Daher muss beim ersten Start vom Router „Firewall“ „Kopiert“ ausgewählt werden. Damit die DHCP-Zuordnungen einwandfrei funktionieren wird empfohlen, die VMs in folgender Reihenfolge hochzufahren:

1. Firewall
2. R0
3. R1
4. Server
5. Client S0
6. Client S2
7. Client S1

2.4.6. Einbinden neuer fli4l-Builds

Sollen die Router nun aktualisiert werden, ist einzig das Image in den VM-Ordnern mit dem neuen Build zu ersetzen (z.B. „%USERPROFILE%\Documents\Virtual Machines\R0\fli4l.iso“). Bei diesen Images handelt es sich aktuell um CD-Images. Ein anschließender Neustart der Router-VM lädt die neue Konfiguration. Dabei ist zu beachten, dass der Dateiname des Images „fli4l.iso“ lautet, da die Images in VMware-Player unter diesem Namen statisch eingebunden sind.

3. Infrastruktur

3.1. Einführung

Für die Entwicklung der Router-GUI (Graphical User Interface) muss eine Infrastruktur bestehen, welche die Konfiguration durch einen Client ermöglicht.

Weiterhin sollen statistische Auswertungen die Qualität der Entwicklung auf einem höchst möglichen Niveau halten. Hierzu werden Analysewerkzeuge verwendet, welche beispielsweise den Code auf Programmierfehler prüft oder die Einhaltung der allgemeinen Programmierrichtlinien erleichtern.

Für die so genannte "Continous Integration", ist es notwendig, dass bei bestimmten Ereignissen diese Analysewerkzeuge erneut aufgerufen werden, um eine gleich bleibende Qualität der Infrastruktur zu gewährleisten. Ein solches Ereignis wäre beispielsweise eine Übergabe eines neuen Entwicklungsstandes an das Repository.

3.2. Grundlagen

In diesem Kapitel werden die verschiedenen Werkzeuge beschrieben, die zur Qualitätssicherung in diesem Projekt verwendet werden.

3.2.1. Eclipse

Eclipse ist ein allgemein bekanntes Entwicklungswerkzeug für verschiedene Programmiersprachen und wird in diesem Projektumfeld als bekannt und somit als nicht zu dokumentieren vorausgesetzt. Im Rahmen dieses Projektes wird Eclipse in der Version 4.2.1 (Juno) und das Java SDK in der Version 1.6 verwendet.

Abbildung 6: Eclipse

3.2.2. FindBugs

FindBugs ist ein Programm, das mithilfe einer statischen Analysen Fehler in Java Code finden kann. Diese Analyse weist meistens auf einen tatsächlichen Programmierfehler hin. Statisch bedeutet in diesem Zusammenhang, dass der Bytecode, also der kompilierte Code, des Programms analysiert wird. Dies geschieht ohne die Ausführung des Programms.
Die aktuelle Version 2.0.2 von FindBugs soll in diesem Projekt verwendet werden.

Abbildung 7: FindBugs

3.2.3. PMD

Abbildung 8: PMD

PMD überprüft den Java Code auf Verletzungen bestimmter Programmierrichtlinien. Die Fehler, die PMD findet, sind typischerweise keine echten Fehler, sondern eher ineffizienter Code, d. h. die Software wird in der Regel trotzdem korrekt ausgeführt, wenn die Verstöße nicht korrigiert werden. PMD findet auf Basis von statischen Regeln potentielle Probleme wie beispielsweise:

- Mögliche Bugs – Leere try/catch/finally/ switch-Blöcke.
- Toter Code – Ungenutzte lokale Variablen, Parameter und private Methoden
- Leere if/while-Anweisungen
- Überkomplizierte Ausdrücke – Unnötige if-Anweisungen, oder for-Schleifen, die stattdessen als while-Schleifen genutzt werden könnten
- Suboptimaler Code – verschwenderische Nutzung von String/StringBuffer
- Klassen mit hoher Zyklomatischer Komplexität [8]

PMD bietet eine Vielzahl eingebauter Regeln und die Möglichkeit, eigene Regeln zu schreiben. Im Java Umfeld werden üblicherweise neben PMD auch Checkstyle und FindBugs für denselben Zweck eingesetzt.

In diesem Projekt soll die aktuelle Version des Eclipse Plugins 3.2.6 verwendet werden. Auf dem Server wird aus Kompatiblitätszwecken die Version 4.2.6 genutzt.

[8] Die "zyklomatische Komplexität nach McCabe“ gibt die Anzahl der Pfade wieder, welche durch ein Softwaremodul durchlaufen werden. Mathematisch ausgedrückt berechnet sich die Komplexität durch L - N + 2P. L = Anzahl der Kanten eines Kontrollflussgraphen N = Anzahl der Knoten eines Kontrollflussgraphen P = Anzahl der Verbundkomponenten eines Kontrollflussgraphen.

3.2.4. Checkstyle

Checkstyle ist ein Tool, das oft in der kooperativen Softwareentwicklung verwendet wird, um einen einheitlichen Programmierstil zu wahren. Es nutzt eine statische Code-Analyse, die den Quelltext zu Grunde legt. Durch den Einsatz von Checkstyle werden die einheitliche Lesbarkeit, die Softwarequalität und die Wiederverwendbarkeit des Programmcodes verbessert. Es ist jedoch kein Werkzeug zur automatisierten Erkennung von Programmierfehlern oder zur automatischen Behebung eines fehlerhaften Programmierstils. Es kann jedoch die in Eclipse integrierte automatische Formatierung genutzt werden, um einige Fehler beheben.

Abbildung 9: Checkstyle

Checkstyle wurde im Jahr 2001 von Oliver Burn entwickelt und wird seitdem von einer Reihe von Entwicklern als OpenSource Software weiterentwickelt und gewartet.

Die aktuelle Version 5.6 soll innerhalb dieses Projektes verwendet werden.

3.2.5. Cobertura

Cobertura ist ein Tool, welches die Testabdeckung in Prozent ermittelt. Es werden nicht geprüfte Codefragmente hervorgehoben, um die Testabdeckung des Codes auf ein höchst mögliches Niveau zu bringen. Die Testabdeckung gibt an, wie viele Teile des geschriebenen Codes tatsächlich getestet wurden. Da JUnit Tests manuell erstellt werden müssen, hängt es ohne Cobertura vom Tester ab, wie hoch die Testabdeckung tatsächlich ist.

Cobertura ist ebenfalls als Eclipse Plugin erhältlich, dieses wird eCobertura genannt. Die aktuelle und von uns genutzte Version ist 0.9.8.

3.2.6. Ant

Das OpenSource Projekt „Ant“ (Another Neat Tool) ermöglicht die automatisierte Erstellung von Softwarepaket. Dies kann auf Basis von Bibliotheken, bereits vorhandenem Quelltext oder anderen Dateien geschehen.

Die Steuerung Ants geschieht durch eine XML- beziehungsweise Build-Datei. Diese Datei gilt als Fundament für das zugrundeliegende Projekt.

Die aktuelle Version ist 1.8.4.

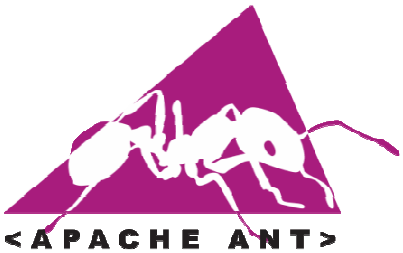


Abbildung 10: Ant

3.2.7. Subversion

Subversion (SVN) ist eine freie Software zur Versionsverwaltung von Dateien und Verzeichnissen. Subclipse ist ein Subversion Client für Eclipse und wird im Rahmen dieses Projektes in der Version 1.0.0 verwendet.

Mit Subversion kann ein Team gemeinsam an einem Projekt arbeiten, das in einem zentralen Projektarchiv (Repository) liegt. Subversion wacht über Änderungen der Dateien und weißt auf eventuelle Fehler hin. Wenn beispielsweise zwei Entwickler an der gleichen Datei arbeiten und den selben Versionsstand ausgecheckt haben, merkt Subversion dies und hilft bei der Zusammenführung der Änderungen, damit sich die beiden Entwickler nicht versehentlich die Dateien überschreiben.

Abbildung 11: Subversion

Wenn Änderungen an Inhalten verteilt auf den Computern der Entwickler ausgeführt werden, werden zwischen dem Projektarchiv und einem Arbeitsplatz jeweils nur die Unterschiede zu bereits vorhandenen Ständen übertragen.

3.2.8. Jenkins

Jenkins wird zur kontinuierlichen Integration von Softwareentwicklungen genutzt. Hauptsächlich wurde dies von Kohsuke Kawaguchi, einem Mitarbeiter von Sun Microsystem, unter dem Namen Hudson entwickelt. Als Sun von Oracle aufgekauft wurde, verließ er das Unternehmen und benannte sein Projekt in Jenkins um.

Abbildung 12: Jenkins

Jenkins ist in Java geschrieben und läuft in einem beliebigen Servlet-Container. Es muss jedoch keiner installiert werden, da ein eigener namens Winstone mitgebracht wird. Dank der Unterstützung von verschiedenen Build-Tools wie Apache Ant, und Versionsverwaltungssystemen wie CVS oder Subversion, können sehr gut Java, PHP und .NET.basierte Programme in der Entwicklung unterstützt werden. Dies geschieht, zum Beispiel durch die Integration von Testtools wie JUnit.

3.2.9. XML Merge

XML Merge ermöglicht eine verlustfreie Zusammenfassung von mehreren XML Dateien zu einer Einzigen. Somit können die Auswertungen von Checkstyle auf den normalen Sourcen und den Testklassen zu einer einzigen Auswertung verbunden werden.

3.3. Einrichtung und Verwendung der Werkzeuge in Eclipse

In diesem Kapitel wird die Installation, Einrichtung und Verwendung der einzelnen Werkzeuge in Eclipse beschrieben.

3.3.1. FindBugs

Installation

8. Plugin über „install new Software“ von der Seite der Entwickler [9] installieren

9. Die Konfiguration (.fbprefs) für Findbugs wird automatisch aus dem Repository genutzt.

10. Wenn die Analyse durch FindBugs automatisch durchgeführt werden soll, so muss unter „Properties“ → „FindBugs“ der Haken bei „Run automatically“ gesetzt werden.

Verwendung

Um eine Prüfung durchzuführen, muss im Kontextmenü des Package Explorer die Funktion „Find Bugs“ → „Find Bugs“ verwendet werden. Um zu den gefundenen Fehlern zu gelangen, muss der Benutzer in die Find Bugs Perspektive umgeschaltet, in der alle Bugs auflistet werden.

[9] http://findbugs.cs.umd.edu/eclipse

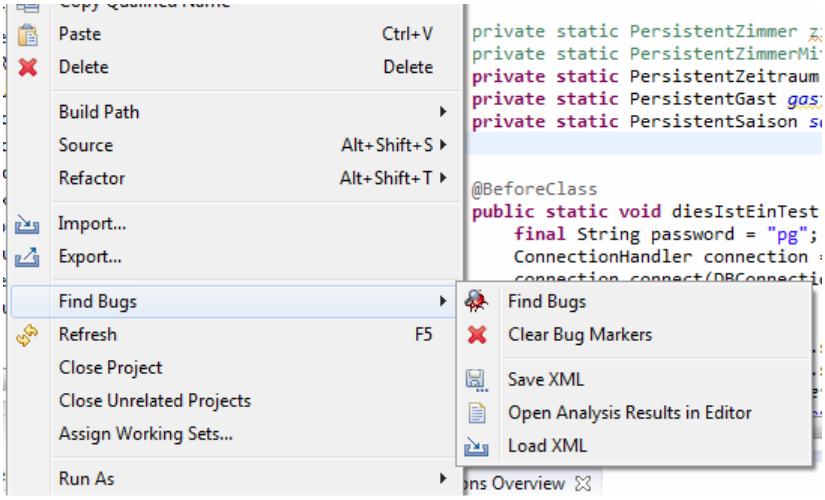


Abbildung 13: Verwendung Findbugs

Um das Plugin wieder auszuschalten und die Bugs zu löschen kann im Kontextmenü von Find Bugs der Punkt „Clear Bug Markers“ ausgewählt werden. Die Punkte „Save XML“ sowie „Load XML“ ermöglichen dem Benutzer die Analyseergebnisse als XML-Datei zu exportieren sowie zu importieren. Dieses Analyseergebnis kann vom Benutzer zudem im Editor geöffnet werden („Open Analysis Result in Editor“).

3.3.2. PMD

Installation

1. Plugin über „install new Software“ von der Seite der Entwickler [10] installieren.
2. Die Konfiguration (.ruleset) für PMD wird ebenfalls automatisch aus dem Repository genutzt.
3. Soll die Analyse durch PMD automatisch durchgeführt werden, so muss unter „Properties“ → „PMD“ der Haken bei „Enable PMD“ gesetzt werden.

Verwendung

Um eine Prüfung manuell durchzuführen, muss im Kontextmenü des Package Explorers die Funktion „PMD“ → „Check Code With PMD“ verwendet werden. Dadurch wird automatisch in die PMD Perspektive umgeschaltet, in der alle Verletzungen auflistet werden.

[10] http://pmd.sourceforge.net/eclipse

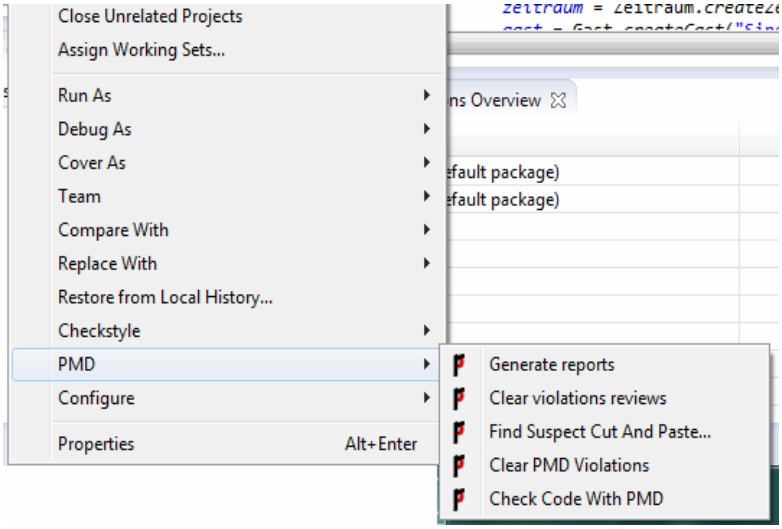


Abbildung 14: Verwendung PMD

Im Kontextmenü von PMD kann außerdem ein Protokoll erstellt werden („Generate reports“), alle angezeigten Verstöße ausgeblendet werden („Clear violations reviews“), kopierter Quelltext gefunden werden („Find Suspect Cut And Paste“) oder auch die gefundenen PMD Verstöße gelöscht werden („Clear PMD Violations“).

3.3.3. Checkstyle

Installation

1. Plugin über „install new Software“ von der Seite der Entwickler installieren [11]

2. Auch hierbei wird die Konfiguration über das Repository automatisch verteilt und muss durch den Anwender nicht angepasst werden.

Verwendung

Ist Checkstyle erfolgreich für das Projekt konfiguriert, gibt es die Möglichkeit Checkstyle für ein Projekt zu aktivieren beziehungsweise zu deaktivieren. Hierzu öffnet der Benutzer das Kontextmenü des Projekts und wählt unter dem Reiter „Checkstyle“ → „activate Checkstyle“ aus (siehe Abbildung 10).

Abbildung 15 - Verwendung Checkstyle

Ist Checkstyle für ein Projekt aktiviert worden, kann nun das Projekt auf Verletzungen bezüglich der Konfiguration untersucht werden. Diese Verletzungen werden folglich tabellarisch angegeben (Siehe Abbildung 11). In dieser Tabelle werden die Verletzungen nach Art kategorisiert und ihre Position dargestellt. Werden die Verletzungen bereinigt („Clear Checkstyle violations“) , verschwinden diese in der Auflistung automatisch.

[11] http://eclipse-cs.sourceforge.net/update

Neben einem Projekt können auch einzelne Klassen durch Checkstyle untersucht werden.

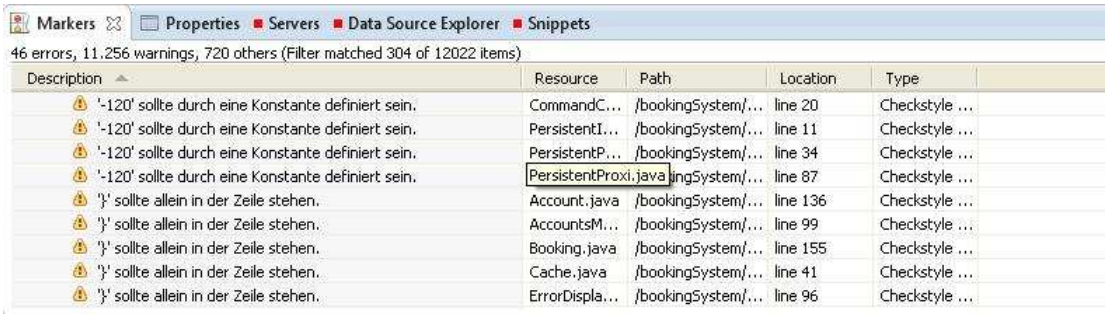


Abbildung 16: Checkstyleanzeige

3.3.4. Cobertura

Installation

Plugin über „install new Software“ von der Seite der Entwickler12 installieren. Dies funktioniert ohne Clientkonfiguration

Verwendung

Die Verwendung Coberturas beziehungsweise eCoberturas ist leicht, da dies ein eher im Hintergrund agierendes Werkzeug ist.

Um die Testabdeckung zu erfahren, muss der Benutzer den JUnit-Test unter dem Punkt „Cover as“ auswählen. Nun wird der Test regulär ausgewertet.

Zusätzlich zur Konsolenausgabe kann eine so genannte „Coverage Session View“ in Eclipse geöffnet werden, um detailliertere Informationen zu den einzelnen Tests zu erhalten.

Abbildung 17: Verwendung Cobertura

[12] http://ecobertura.johoop.de/update

3.3.5. Subversion (Subclipse)

Installation

Plugin über „install new Software“ von der Seite der Entwickler [13] installieren. Dies funktioniert ohne Clientkonfiguration.

Nun muss das Repository nur noch eingerichtet werden.

Einrichtung

Zu Beginn muss die Repository-View aufgerufen werden.

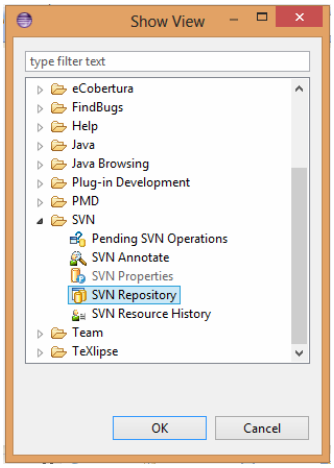


Abbildung 18: Repository View aufrufen

Anschließend soll eine neue Location mit folgender URL eingerichtet werden: https://fhdwdev.ha.bib.de/svn/ipfli4l13/trunk.

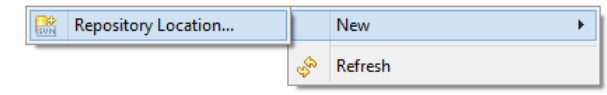


Abbildung 19: Repository Location hinzufügen

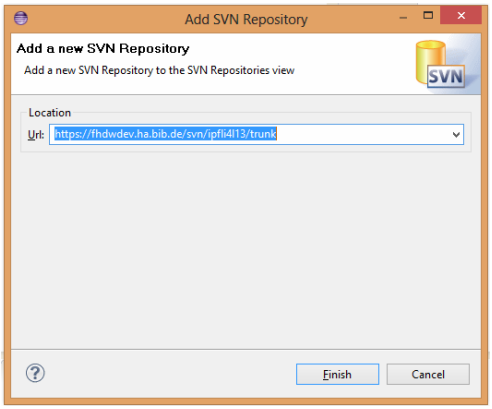


Abbildung 20: Repository Location

[13] http://subclipse.tigris.org/update

Die Anmeldedaten sind hfw410XX und euer Passwort aus der Domäne. Das Auschecken des Projektes funktioniert wie in Abbildung 16 beschrieben. Hinterher muss ein Name für das Projekt vergeben werden.

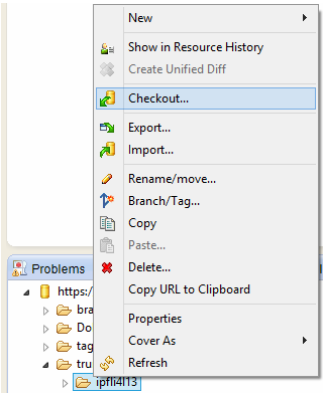


Abbildung 21: Checkout Projekt

Nun kann das Projekt genutzt werden.

Die Dokumentationen können auch eingecheckt werden. Die URL hierfür ist https://fhdwdev.ha.bib.de/svn/ipfli4l13/Dokumentation.

Verwendung

Nun ist das Projekt aus dem Repository im Package Explorer und kann wie ein lokalen Projekt verwendet werden. Wenn die Arbeit abgeschlossen ist, sollte das Projekt zurück ins Repository geschrieben werden. Dies funktioniert mit dem Kontextmenü auf dem Projekt. Vor jedem Commit der Daten sollte das Repository synchronisiert („Synchronize with Repository“) werden.

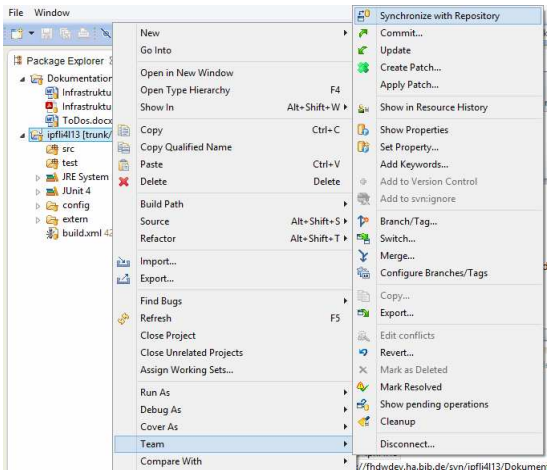


Abbildung 22: Synchronisieren der Daten

Hiermit öffnet sich eine neue Ansicht, die die Veränderungen im Repository anzeigt. Dort können dann sowohl die Daten dem Repository hinzugefügt werden („Commit“) als auch die bereits geänderten Daten zusammengefügt („Merge“) oder erneuert werden („Update“). Über das Kontextmenü des Projektes oder auch auf einzelnen Dateien können diese Funktionen aufgerufen werden.

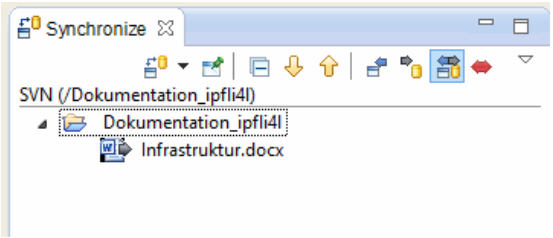


Abbildung 23: Änderungen im Repository

3.4. Aufbau und Funktion des Ant-Skripts

Ein Build wird den Einstellungen nach automatisiert jeden Morgen um 3 Uhr aufgerufen. Bei jedem Build wird ein Ant-Skript ausgeführt. Dieses heißt build.xml und liegt direkt im Projektverzeichnis Ipfli4l13 im Repository. Das Buildskript kompiliert zunächst den Java Quellcode und führt anschließend die oben genannten Werkzeuge zur Analyse der Softwarequalität aus.

Begonnen wird mit dem Task „Javac“. Hier werden zunächst alle Java Sourcen mit dem Java Compiler in Bytecode umgewandelt. Im Anschluss wir die Javadoc erstellt.

Als nächstes folgt der Task „Testc“. Hier werden alle Testklassen kompiliert. Die Kompilierung der Testklassen setzt voraus, dass der normale Sourcecode erfolgreich kompiliert werden konnte. Wenn die Testklassen erfolgreich kompiliert wurden, werden die Testklassen von Cobertura instrumentiert. Im Anschluss werden die JUnit-Tests ausgeführt und protokolliert. Dies geht natürlich nur, wenn die vorherige Kompilierung der Testklassen erfolgreich war. Die Ergebnisse der JUnit Tests, befinden sich nach Ablauf des Buildprozesses im Ordner reports/junit, die von Cobertura im Ordner reports/coverage.

Nun wird Checkstyle aufgerufen. Hierbei ist wichtig, dass, genau wie in Eclipse, die normalen Sourcen und die Testklassen mit unterschiedlichen Vorgaben geprüft werden. Zunächst werden die normalen Sourcen mit Hilfe von Checkstyle analysiert. Als Vorgabe wird hierzu die Datei checkstyle.xml verwendet, die ebenfalls vom Eclipse Plugin genutzt wird. Im Anschluss werden die Test Sourcen analysiert. Hierzu wird die Datei checkstyle-test.xml verwendet, die ebenfalls vom Eclipse Plugin zur Analyse der Testklassen genutzt wird. Als nächstes werden die Ausgabedateien beider Analysen zu einer gemeinsamen zusammengefasst. Dies geschieht mit Hilfe von XML Merge. Die Ergebnisse der Checkstyle-Analyse befinden sich nach dem Buildprozess im Ordner reports/checkstyle.

Nach der Checkstyle-Analyse wird FindBugs aufgerufen. Da dies auf der Bytecodeebene durchgeführt wird, wird für diesen Task vorausgesetzt, dass die Kompilierung der normalen Sourcen und der Testklassen fehlerfrei verlaufen ist. Auch für FindBugs wird dieselbe Konfigurationsdatei verwendet, wie vom Eclipse Plugin. Die Ergebnisse der FindBugs-Analyse befinden sich nach dem Buildprozess im Ordner reports/findBugs.

Als letztes wird der Quellcode mit Hilfe von PMD überprüft. Wie bei den anderen Analysewerkzeugen auch, wird auch hier dieselbe Konfiguration wie in Eclipse verwendet. Nach Beenden des Buildprozesses befindet sich das Ergebnis der PMD-Analyse im Ordner reports/pmd.

3.5. Einrichtung von Jenkins

Die Einrichtung des Jenkins Projekts ist durch die intuitiv geführte Weboberfläche ohne größere Probleme möglich. Die Erstellung eines neuen Projektes wird aufgrund von Rechtemangel durch einen Administrator auf dem Server erledigt. Die weitere Konfiguration des Projektes liegt jedoch in unseren Händen.

Zu Beginn wird das eingerichtete Quellcodeverwaltungssystem an Jenkins angebunden. Hierbei ist zu beachten, dass die Sourcen bei jedem Build aktualisiert werden. Ansonsten könnte es sein, dass ein neu gestarteter Build mit alten Sourcen durchlaufen wird, was natürlich nicht passieren soll.

Abbildung 24: Source-Code-Management

Hinterher wird das bereits fertige Ant-Skript eingebunden. Dies wird durch die Angabe des Pfades zum Skript erledigt.

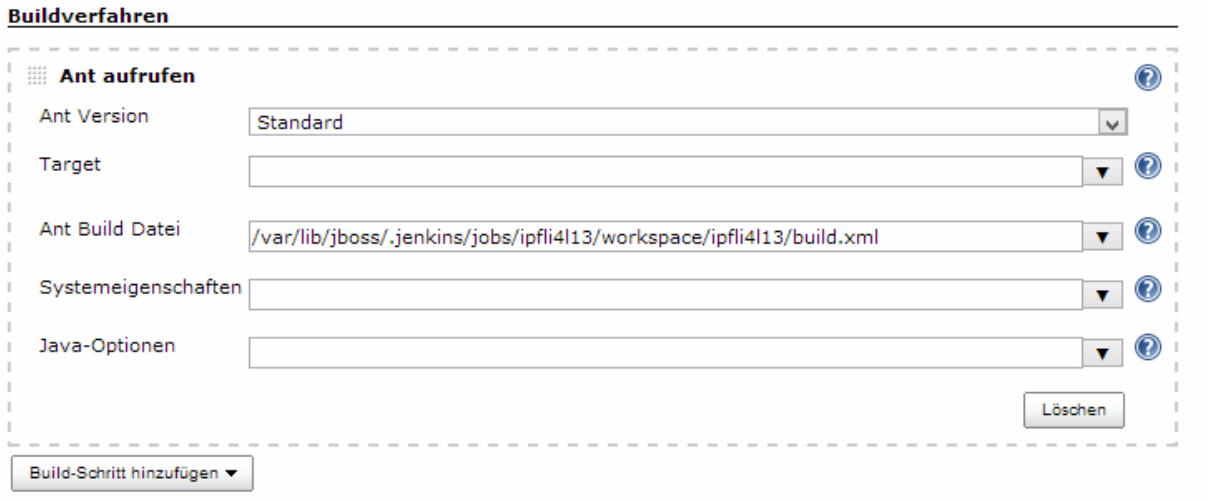


Abbildung 25: Buildverfahren

Im Anschluss folgt die Einrichtung der Post-Build-Aktionen. Es muss jeweils für die Veröffentlichung der Analyseergebnisse von Checkstyle, FindBugs, PMD und Cobertura, der JavaDoc und der JUnit-Testfälle eine Post-Build-Aktion eingerichtet werden.

Bei der Einrichtung des Post Builds für die Veröffentlichung der Checkstyleergebnisse muss der Pfad der XML-Ergebnis-Datei angegeben werden. Diese werden darauf hin intern von Jenkins grafisch aufgearbeitet. Außerdem müssen die Schwellwerte für die Jenkins interne Anzeige festgelegt werden. Diese liegen für 100% bei 0 Fehlern und bei 0% wenn mindestens 10 Fehler aufgetreten sind. Ein instabiler Build liegt bei mehr als 0 Fehlern vor, ein fehlgeschlagener bei mehr als 10.

Bei der Einrichtung des Post-Builds für die Veröffentlichung der FindBugs-Analyse muss auch der Pfad der XML-Ergebnis-Datei angegeben werden. Außerdem müssen die Schwellwerte festgelegt werden. Diese liegen für einen erfolgreichen Build (100%) bei 0 und bei einem fehlgeschlagenen Bild bei mehr als 5 Fehlern. Somit wird der Build bei mehr als 0 Fehlern als instabil und bei mehr als 5 Fehlern als fehlgeschlagen angezeigt.

Die Einrichtung der Veröffentlichung der Ergebnisse der PMD-Analyse erfolgt wie die oben genannten über die Angabe des Pfades der XML-Datei. Hierbei liegen die Werte für den Gesundheitszustand des Build für 100% bei 0 und für 0% bei 7 Fehlern.

Bei den Veröffentlichungseinstellungen der Javadoc wird ebenfalls der Pfad zum Dokument angegeben.

Die Veröffentlichung der JUnit-Testergebnisse wird über die Angabe des Pfades zur XML-Datei konfigurieren.

Die Cobertura Testabdeckung wird auch wie bei den anderen Werkzeugen über eine XML-Datei angegeben. Die Testabdeckungsziele liegen für Verzweigungen bei 70% und für Zeilen und Methoden bei 80%.

3.6. Nutzung von Jenkins

Die Jenkinsinstallation mit der im Rahmen des Fli4l-Projektes gearbeitet werden soll ist unter https://fhdwdev.ha.bib.de/jenkins/ zu finden. Betritt der Benutzer diese Seite, so ist dort eine Liste mit unterschiedlichen Projekten zu sehen.

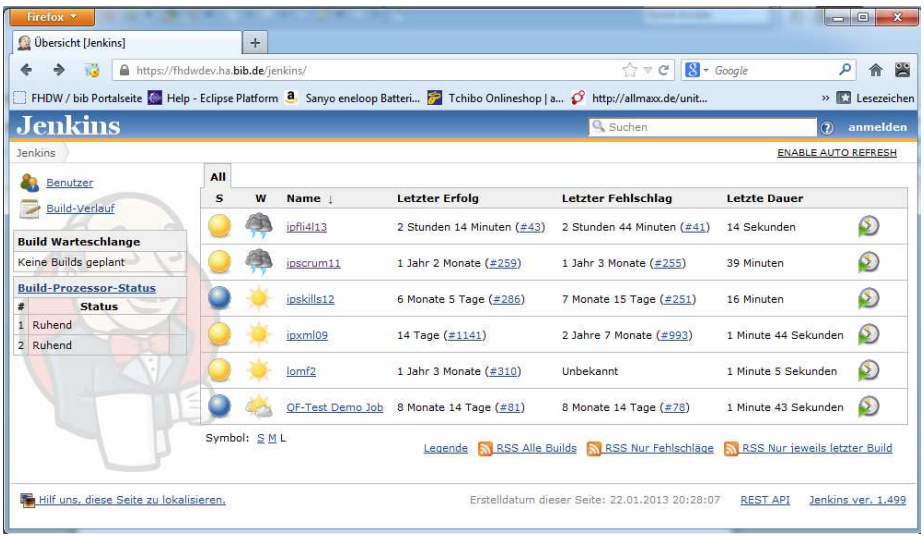


Abbildung 26: Übersicht Jenkins

Nun muss das oberste Projekt (ipfli4l13) ausgewählt werden. Der Benutzer gelangt nun auf eine Seite, auf der er genauere Informationen zum Projekt erhält.

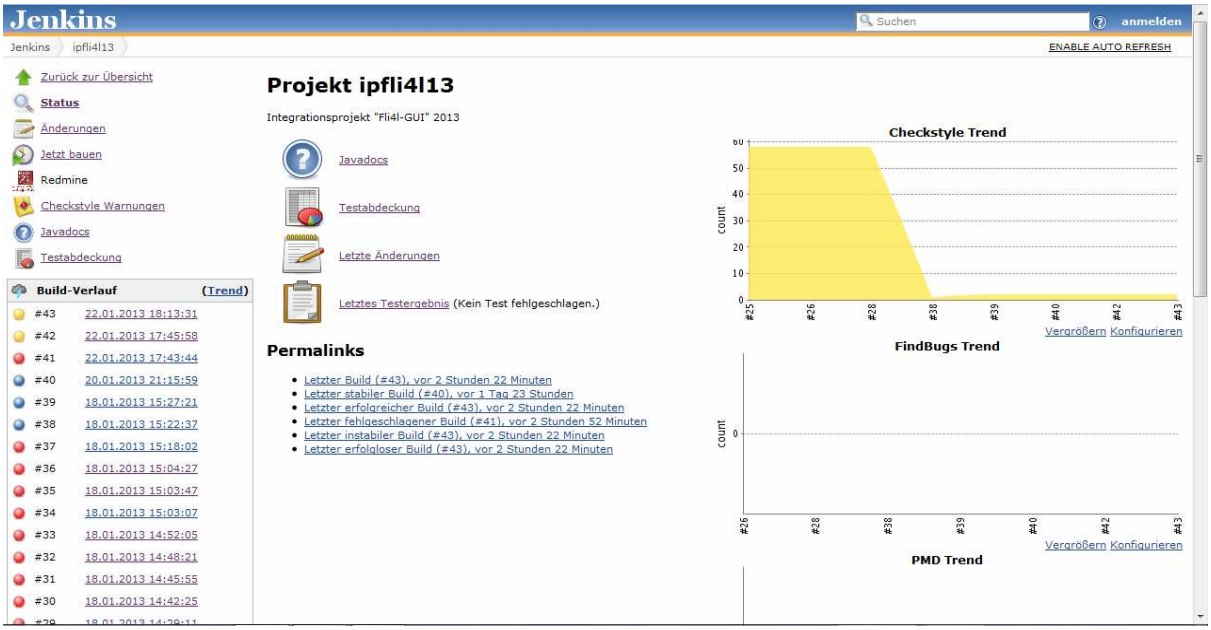


Abbildung 27: Projektansicht

Die Seite ist dabei so aufgebaut, dass sich im rechten Bereich Informationen zu den einzelnen Auswertungen zum Projekt befinden. Hier sieht der Benutzer beispielsweise Statistiken zu den Analysen von Checkstyle, FindBugs, PMD, JUnit und Cobertura.

In der Mitte der Seite befinden sich der Link zur Javadoc, der Link zur detailierten Auswertung der Testabdeckung, der Link zu den letzten Änderungen und ein Link zu den letzten Testergebnissen. Auf der linken Seite unten ist der Build-Verlauf zu finden. Hier sind die letzten Builds mit den jeweiligen Startzeiten aufgeführt. Ein farbiger Punkt vor dem Build kennzeichnet, ob er Build erfolgreich, instabil oder fehlgeschlagen ist. Über „Trend“ kann eine Statistik der letzten Builds aufgerufen werden.

Auf der linken Seite im oberen Bereich befindet sich weitere Links zu Informationen über das Projekt. Hier befinden sich unter anderem noch einmal die Javadoc und die Testabdeckung. Es gibt aber auch Informationen zu den Checkstyle Warnungen und den Änderungen. Sehr wichtiger ist hier der Punkt „Jetzt bauen“. Wird dies ausgewählt, so wird ein neuer Build angestoßen. Mit „Zurück zur Übersicht“ gelangt der Benutzer wieder auf die Übersicht aller Projekte.

Wurden oben links die Änderungen ausgewählt, so gelangt der Benutzer zu einer Übersicht aller Änderungen, die im Laufe der Zeit am Projekt durchgeführt worden sind. Hierbei ist wichtig, dass jeder Commit kommentiert worden ist, denn diese Kommentare tauchen nun als Beschreibung der Änderungen auf.

Abbildung 28: Änderungen am Repository

Hat der Benutzer die Checkstyle-Warnungen ausgewählt, so öffnet sich eine Seite, auf der die Checkstyle Ergebnisse für jede einzelne Klasse dokumentiert sind. Durch Auswählen einer Klasse, gelangt der Benutzer zu einer detaillierten Beschreibung der in der Klasse aufgetretenen Checkstyle-Warnungen. Durch die Reiter unter der Überschrift „Details“ hat der Benutzer außerdem die Möglichkeit die Warnungen nach verschiedenen Kriterien, wie beispielsweise den Dateien, den Kategorien, und den einzelnen Warnungstypen auszuwählen.

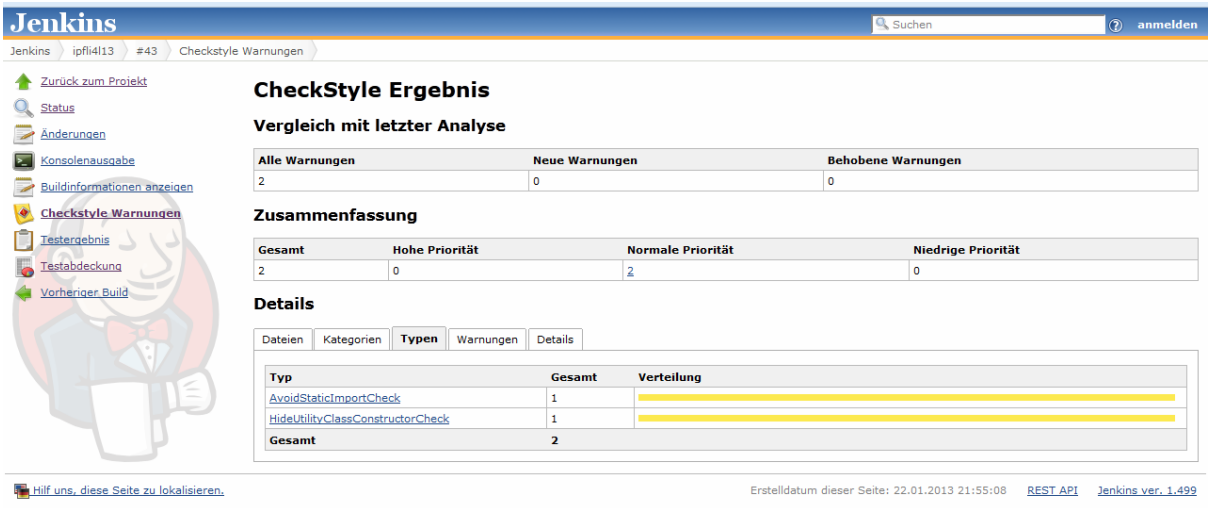


Abbildung 29: CheckStyle Ergebnis

Hat der Benutzer sich für die Testabdeckung entschieden, so öffnet sich eine detaillierte Auswertung der Testabdeckung. Es können verschiedene Abdeckungen, wie beispielsweise die Klassenabdeckung, die Zweigabdeckung, die Pfadabdeckung und einige weitere betrachtet werden. Durch Auswahl der Packages können außerdem die darin liegenden Klassen jeweils einzeln betrachtet werden.

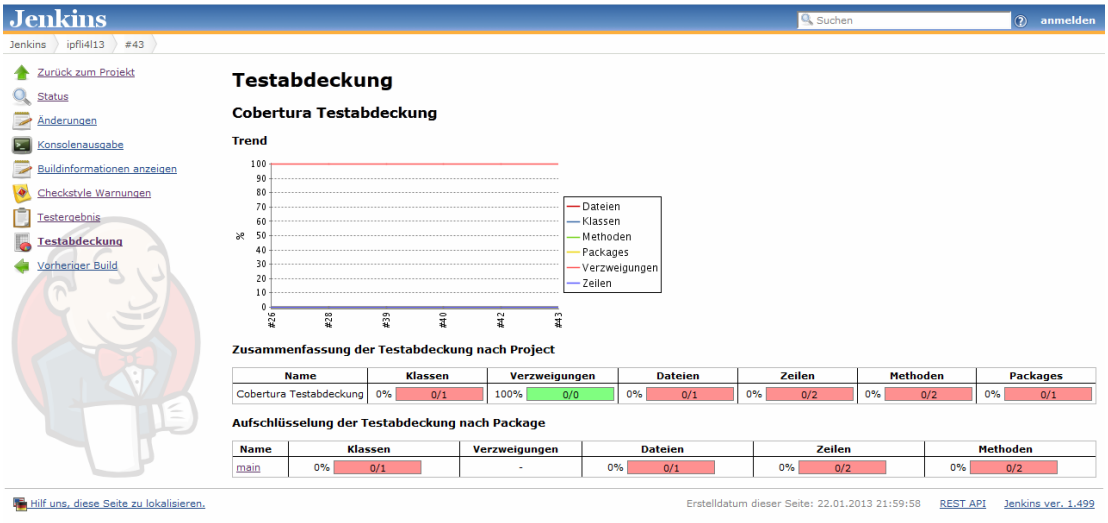


Abbildung 30: Testabdeckung

Zurück zur Projektübersicht gelangt der Benutzer über den Link „Status“ auf der oberen linken Seite. Nun hat er außerdem die Möglichkeit im Build-Verlauf einen Build direkt anzuklicken und so zu einer detaillierten Übersicht des Builds zu gelangen.

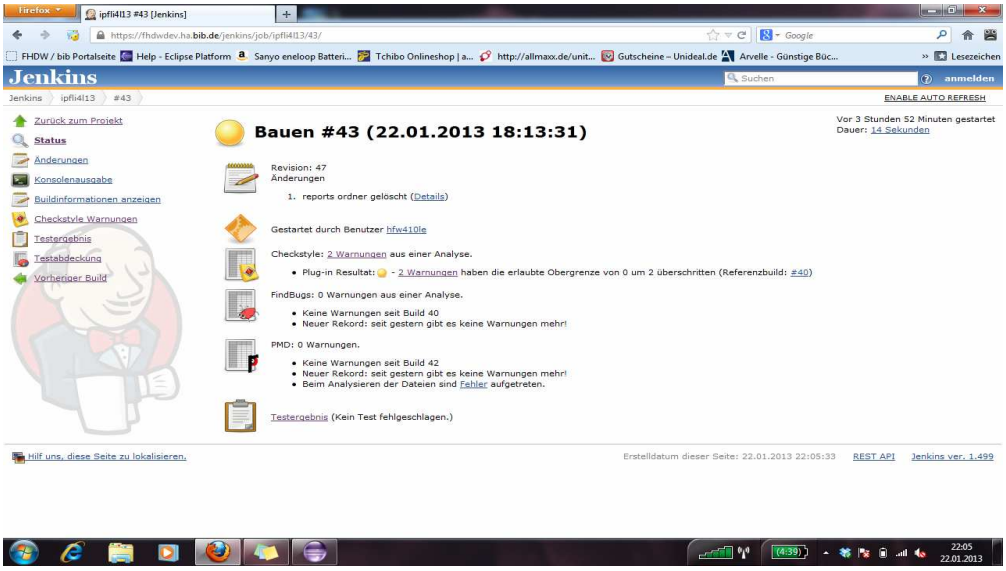
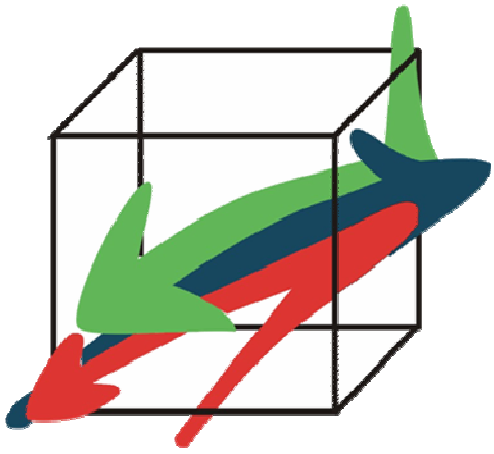

Jenkins
Bauen #43 (22.01.2013 18:13:31)
Zurück zum Projekt
Status
Änderungen
Konsolenausgabe
Buildinformationen anzeigen
Checkstyle Warnungen
Testergebnis
Testabdeckung
Vorheriger Build
Revision: 47
Änderungen
1. reports ordner gelöscht (Details)
Gestartet durch Benutzer hfw410le
Checkstyle: 2 Warnungen aus einer Analyse.
Plug-in Resultat: - 2 Warnungen haben die erlaubte Obergrenze von 0 um 2 überschritten (Referenzbuild: #40)
FindBugs: 0 Warnungen aus einer Analyse.
Keine Warnungen seit Build 40
Neuer Rekord: seit gestern gibt es keine Warnungen mehr!
PMD: 0 Warnungen.
Keine Warnungen seit Build 42
Neuer Rekord: seit gestern gibt es keine Warnungen mehr!
Beim Analysieren der Dateien sind Fehler aufgetreten.
Testergebnis (Kein Test fehlgeschlagen.)
Hilf uns, diese Seite zu lokalisieren.
REST API
Jenkins ver. 1.499

Abbildung 31: Buildstatus

Hier sind genaue Informationen zum ausgewählten Build zu finden, wie beispielsweise die Anzahl der Checkstyle Warnungen, die Anzahl der FindBugs Warnungen, die Anzahl der PMD Warnungen und die Testergebnisse. Außerdem kann er sehen, wer den Build ausgelöst hat und welche Änderungen zuvor durchgeführt wurden. Auf der rechten oberen Seite hat der Benutzer außerdem die Möglichkeit die Konsolenausgabe aufzurufen, die Buildinformationen anzuzeigen und auch hier hat er die Möglichkeit die Testergebnisse, die Testabdeckung und die Checkstylewarnungen für den Build anzuschauen. Möchte er von einer dieser Ansichten wieder zurück zu den allgemeinen Informationen über den Build, so kann er dies über den Link „Status“ erreichen. Will der Benutzer nun wieder zur Projektansicht zurück, so kann er über den Link „Zurück zum Projekt“ dorthin gelangen. Er hat auch die Möglichkeit über „vorheriger Build“ Informationen über den vorher gegangen Build zu bekommen.

Teil 3 – Modelle und Funktionalitäten



Übersicht

Dieser Teil des Handbuchs befasst sich mit der detaillierten Beschreibung der Struktur, in die die grafische Benutzeroberfläche des fli4l-Routers eingebettet ist, sowie den einzelnen Funktionalitäten der Oberfläche.

Einen Überblick über die einzelnen Bereiche der Modellwelt bietet Kapitel 4.

Die grafische Oberfläche erfordert eine korrekte Behandlung der einzelnen Konfigurationsdateien, um die Konfigurationen einzulesen, behandeln und speichern zu können. Da die fli4l-Software in Pakete unterteilt ist, die die Konfigurationsdateien enthalten, wird diese Struktur in einem objektorientierten Klassenmodell widergespiegelt (siehe Kapitel 5).

Das Verarbeiten der verschiedenen Dateien erfolgt durch einen Scanner, einen Parser und einen Printer, die gemäß des Klassenmodells die entsprechenden Objekte anlegen und zur weiteren Bearbeitung bereitstellen (siehe Kapitel 6 – Scanner, Kapitel 7 – Parser, Kapitel 8 – Printer).

Die Struktur und Behandlung der Paketbeschreibungsdateien wird in Kapitel 9 erläutert.

Das Einlesen, Anzeigen und Speichern der fli4l-Konfigurationen wird über eine GUI durchgeführt, die in das Modellkonzept des fli4l-Routers eingepasst wurde und dem Benutzer die entsprechenden Funktionen bereitstellt (siehe Kapitel 10).

4. Klassenmodelle

4.1. Einführung

Zum Überblick über die einzelnen Bereiche der Modellwelt des fli4l-Routers wird im Folgenden ein gesamter Überblick in Form eines Klassenmodells gezeigt. Der Aufbau der einzelnen Bereiche wird in den Folgekapiteln detailliert vorgestellt und erläutert.

4.2. Klassenmodell - Gesamt

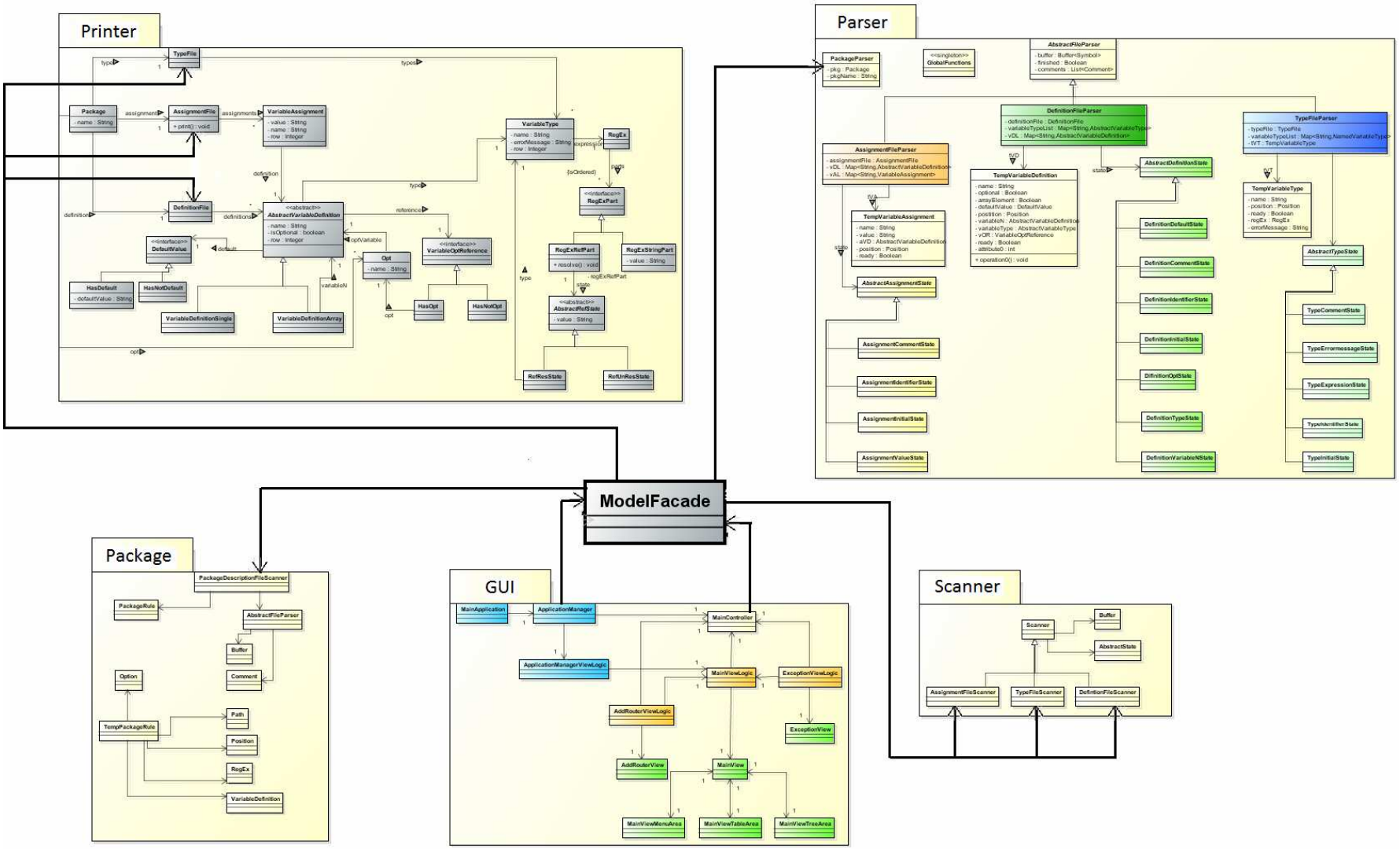


Abbildung 32: Gesamtes Klassenmodel

4.3. Klassenmodell – Scanner

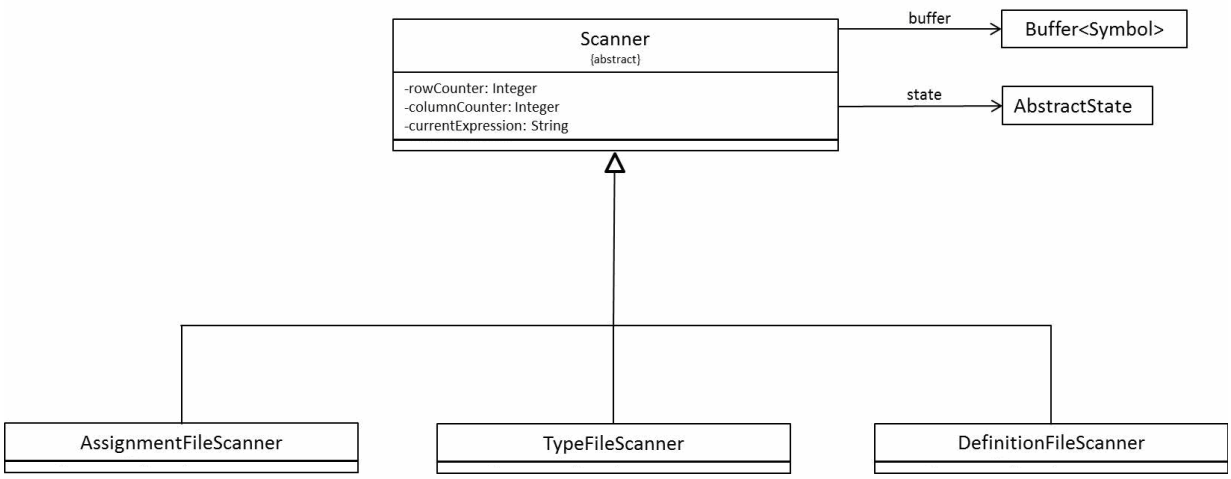


Abbildung 33: Scanner

4.4. Klassenmodell – Parser

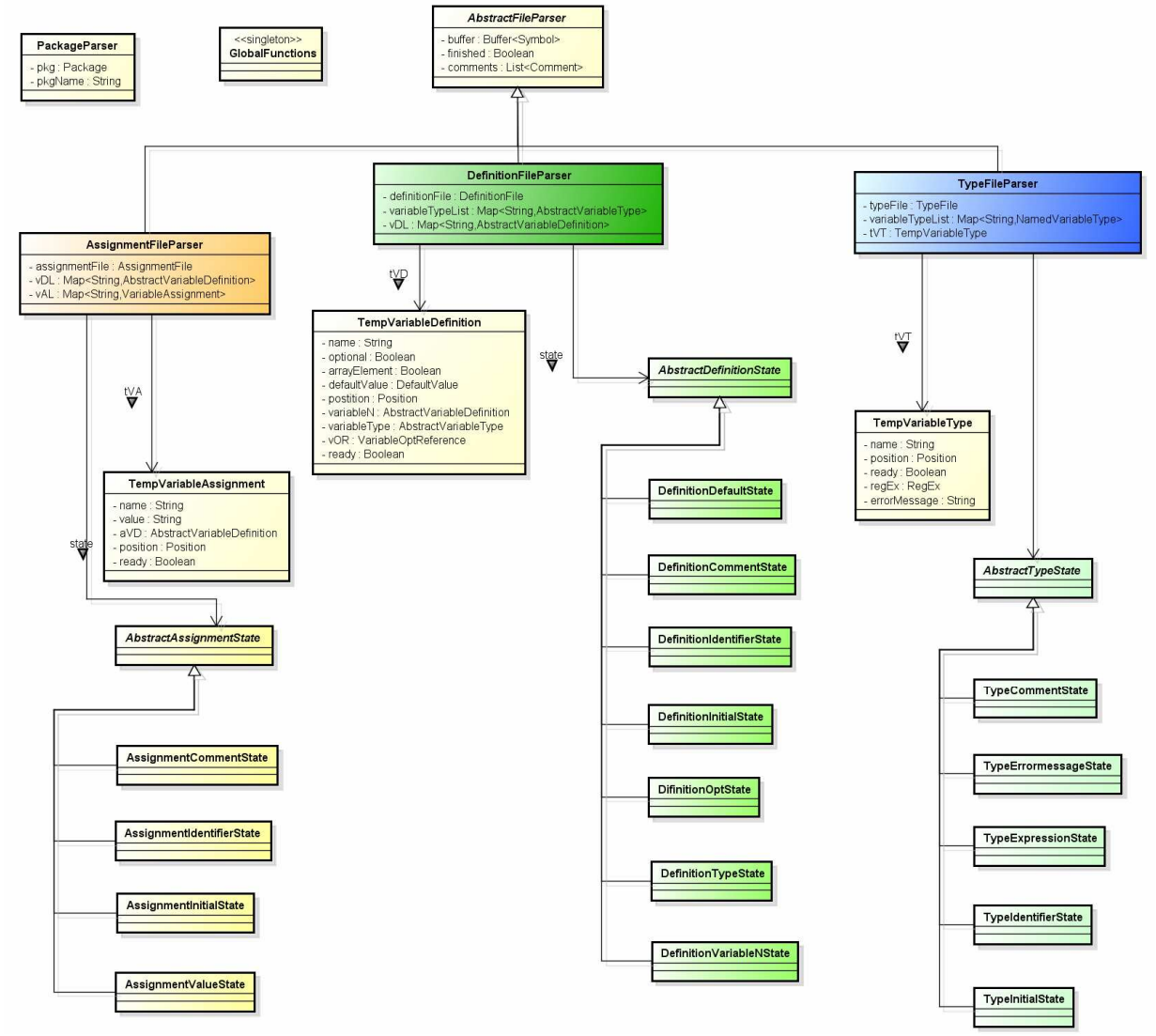


Abbildung 34: Parser

hier wird nicht nurder F

4.5. Klassenmodell - Printer

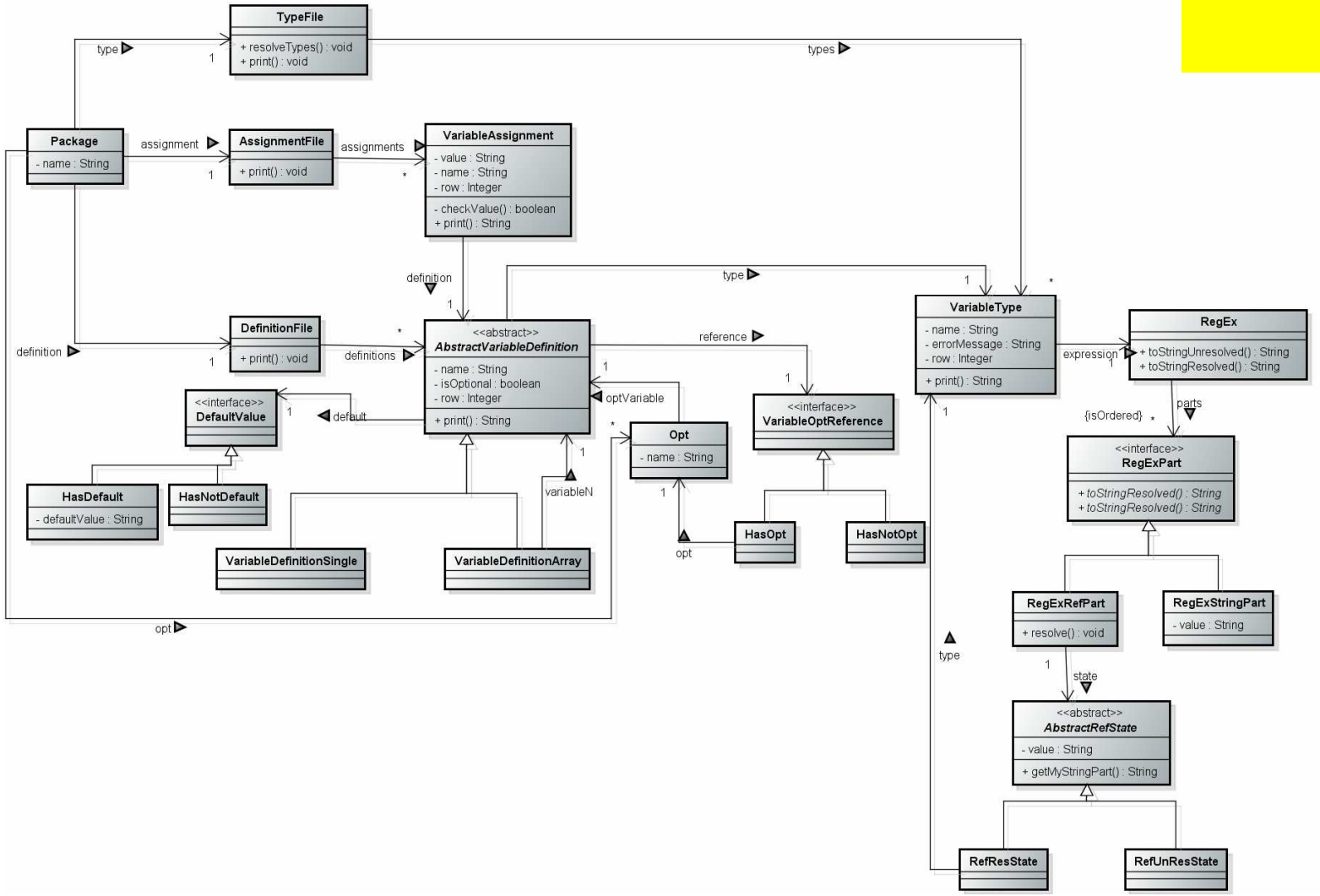
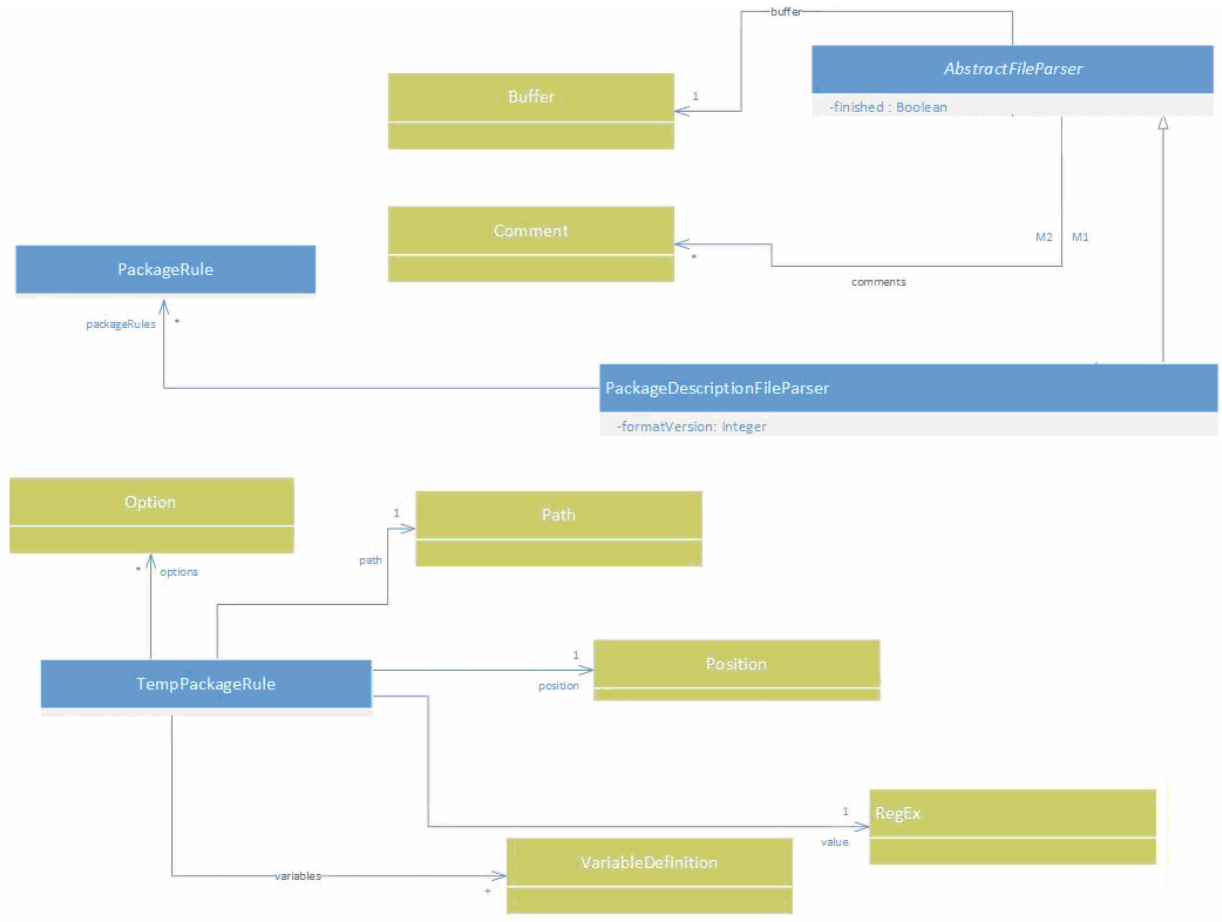


Abbildung 35: Printer

4.6. Klassenmodell – Pakete

Abbildung 36: Pakete



4.7. Klassenmodell - GUI

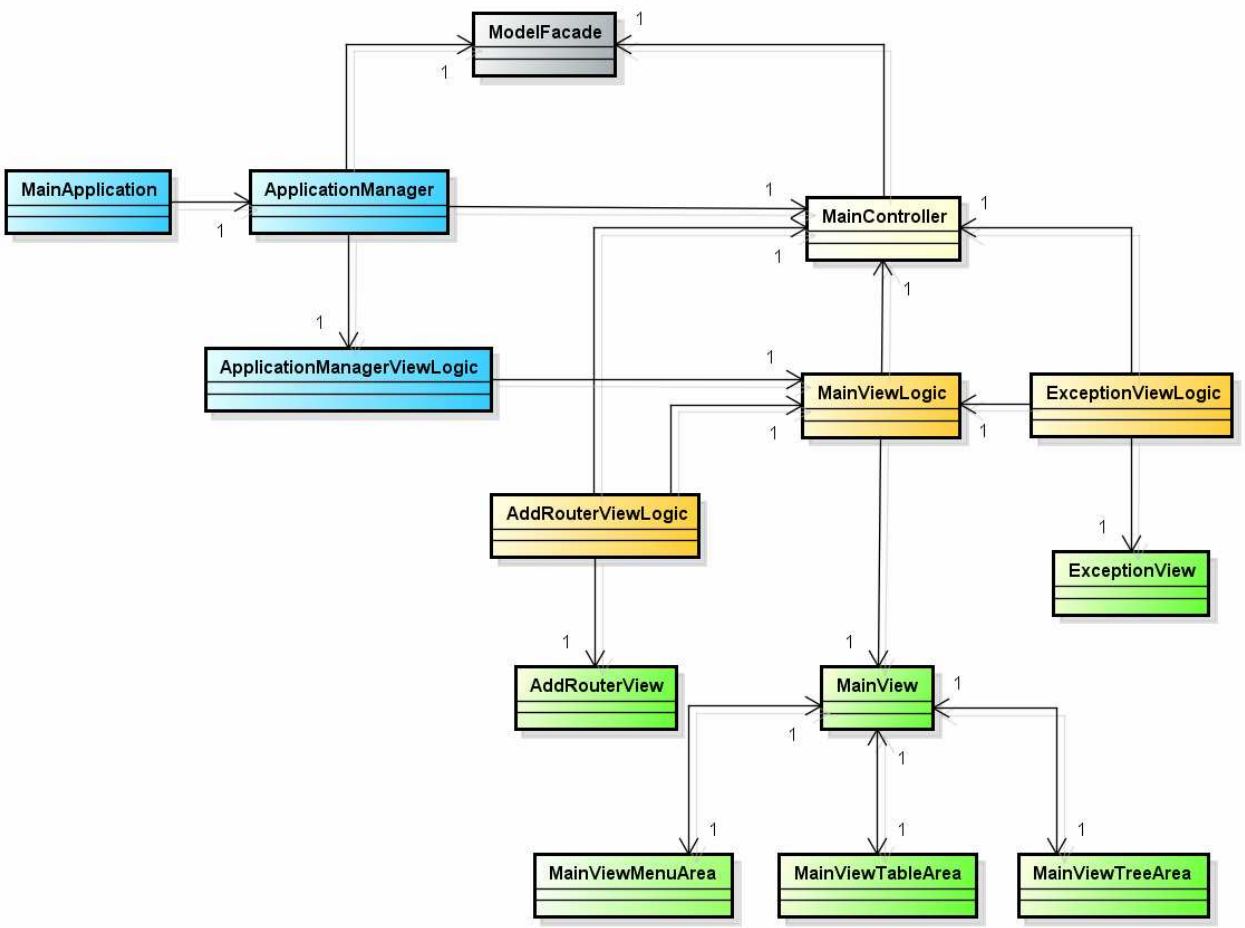


Abbildung 37: GUI

5. Konfiguration

5.1. Einführung

Im Folgenden wird das objektorientierte Modell der fli4l-Konfiguration vorgestellt, das alle relevanten Klassen und Assoziationen der Konfigurationsdateien beinhaltet. Dazu werden zunächst die Konzepte von Variablentyp, Variablendefinition und Variablenbelegung analysiert, die wesentlicher Bestandteil der Paketstruktur der fli4l-Konfiguration sind.

5.2. Paketstruktur

Die fli4l Software ist in mehrere Pakete unterteilt. Für die Basisinstallation ist nur das notwendige „base“-Paket und der jeweilige Betriebssystem-Kernel[14] notwendig. In der Basisinstallation ist lediglich ein Ethernet-Router enthalten. Für beispielsweise DSL, DNS oder WLAN müssen zusätzliche Pakete installiert[15] werden.

[14] Dieser ist für die Analyse der Konfiguration jedoch nicht notwendig und wird in der Dokumentation nicht weiter behandelt.

[15] Eine Auflistung der aktuellen Pakete ist hier zu finden: http://www.fli4l.de/download/stabile-version/pakete/

Neben den beiden Komponenten Basispaket und Zusatzpaket gibt es so genannte optionale Module (OPTs). Einige dieser OPTs befinden sich bereits im Basispaket, andere wiederum können zusätzlich installiert werden.

5.3. Dateistruktur

Das Konfigurieren von fli4l erfolgt paketspezifisch. Es wird über 3 Dateien vorgenommen, die der Variablendefinition, den Variablentypen und den Wertzuweisungen dienen.

- Definitionen: ../check/<Paketname>.txt
- Typen: .. /check/<Paketname>.exp
- Wertzuweisungen: ../config/ <Paketname>.txt

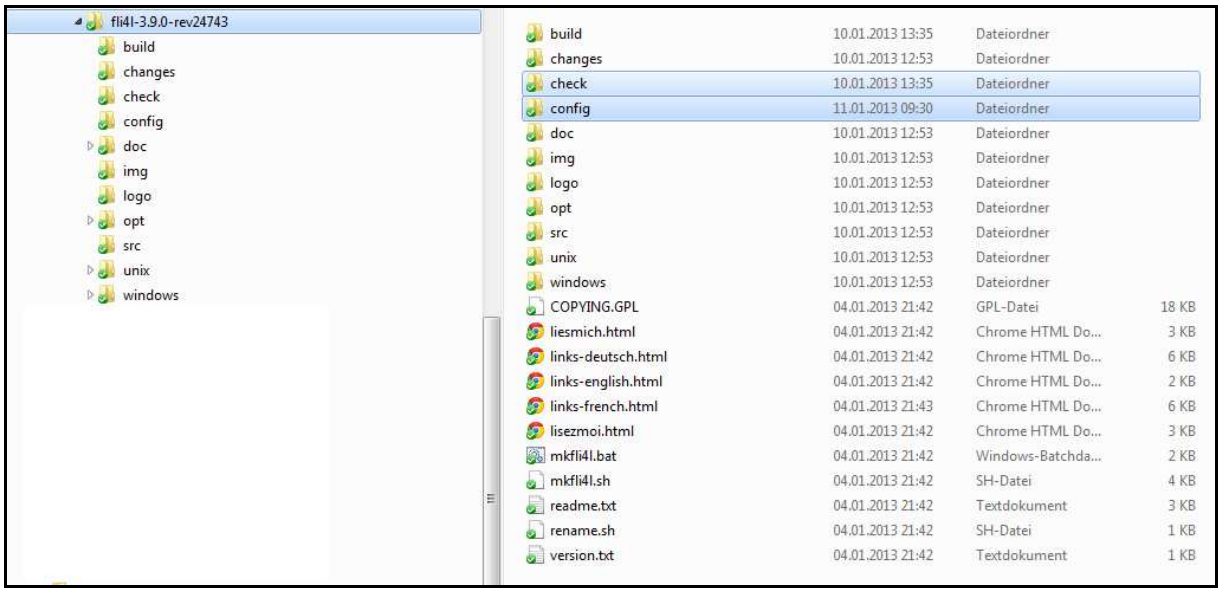


Abbildung 38: Ordnerstruktur fli4l

5.4. Konfigurationsdateien

5.4.1. Variablendefinition / Variable Definition

In der Datei check/<Paketname>.txt werden die Variablen definiert. Dafür wird für jede Variable eine Zeile angelegt. Jede Zeile enthält fünf Spalten:

1. VARIABLE - In der 1. Spalte wird der Name der Variable genannt.

2. OPT_VARIABLE – “Diese Spalte teilt die Variable einem bestimmten OPT zu. Die Variable wird nur auf Gültigkeit überprüft, wenn die hier angegebene Variable auf ’yes’ steht. Gibt es keine Opt-Variable, ist hier ein ’-’ anzugeben.” [Zitat: fli4l-Entwicklerdokumentation, S. 309]

3. VARIABLE_N – Diese Spalte wird momentan ignoriert, da die Informationen anders ermittelt werden. Diese Spalte wird irgendwann entfallen und ist für die Analyse und Umsetzung nicht weiter relevant. Die VARIABLE_N weißt die Arrayelemente einem Array zu.

4. VALUE - Die 4. Spalte gibt an welche Wertzuweisungen für die angegebene Variable legal sind. Entweder wird an dieser Stelle ein Typ genannt oder der Wertebereich direkt durch einen regulären Ausdruck definiert.

5. In der 5. (unbenannten) Spalte kann ein Standardwert angegeben werden.

Des Weiteren besteht die Möglichkeit eine Variable als optional zu kennzeichnen. Dafür wird ein „+“ oder ein „++“ vor den Variablennamen geschrieben. Das „+“ bedeutet, dass diese Variable optional ist. Arrays können statt einem einfachen „+“ auch mit „++“ gekennzeichnet werden. Dieses bedeutet, dass das nachfolgende Array optional ist und dass zusätzlich auch einzelne Elemente des Array fehlen können. In FLI4L werden Arrays mit Hilfe des Prozentzeichens (%) gebildet. Das Prozentzeichen kann dabei in der Mitte des Namens („_%_“) auftreten oder am Ende („_%“) und steht für eine Nummer, die die Position der Variablen im zugehörigen Array anzeigt. Mehrdimensionale Arrays werden durch die Verwendung von mehreren Prozentzeichen im Namen realisiert. Diese können einfach hintereinander („_%_%_“) stehen oder verteilt in einem Wort (Beispiel: PF_USR_CHAIN_%_RULE_%_COMMENT). Wie oben beschrieben werden die einzelnen Arrayelemente in der Variablendefinition durch VARIABLE_N dem Array zugewiesen.

base.txt | base.exp | base.txt

```
1  # Variable            OPT_VARIABLE    VARIABLE_N    VALUE
2  LIBC_VERSION          -               -             NONE "0.9.33"
3  +BUG_CHECK            -               -             YESNO
4  HOSTNAME              -               -             HOSTNAME
5  SER_CONSOLE_IF        SER_CONSOLE     -             NUMERIC
6  SER_CONSOLE_RATE      SER_CONSOLE     -             NUMERIC
7
8  LOG_BOOT_SEQ          -               -             YESNO "yes"
9  NET_DRV_N             -               -             NUMERIC
10 NET_DRV_%             -               NET_DRV_N     NET_DRV
11 NET_DRV_%_OPTION      -               NET_DRV_N     NONE
```

Abbildung 39: Auszug aus check/base.txt (Variablendefinition)

5.4.2. Variablentyp / Variable Type

Die Typen der Variablen werden durch reguläre Ausdrücke spezifiziert. Für die Bildung der regulären Ausdrücke ist die POSIX-Syntax[16] zugrundegelegt. Mit „(RE:NAME)“ kann innerhalb eines Ausdrucks auf andere Ausdrücke verwiesen werden.

Die Wertebereiche (Spalte 4 der Variablendefinition) werden im Regelfall in der check/<Paketname>.exp - Datei mit Hilfe von solchen regulären Ausdrücken definiert. Es ist jedoch auch möglich die regulären Ausdrücke direkt in die check/<Paketname>.txt zu schreiben. Die Syntax für einen Eintrag in der .exp - Datei ist folgende:

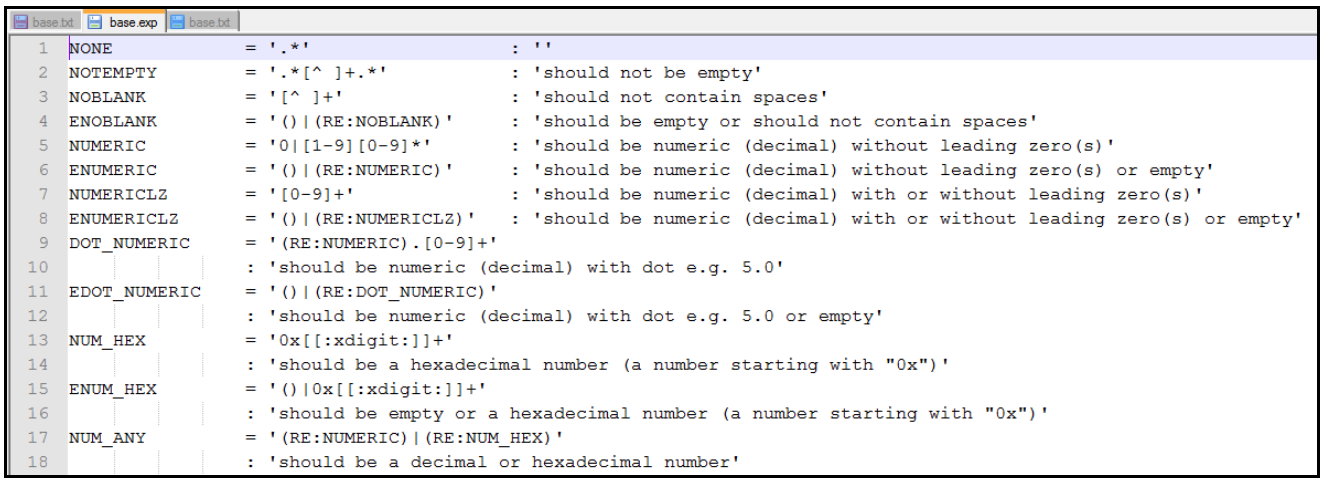
Name = ’Regulärer Ausdruck’ : ’Fehlermeldung’

Exkurs reguläre Ausdrücke: Reguläre Ausdrücke sind Zeichenketten, die der Beschreibung von Sprachen dienen und in der Softwareanwendung Verwendung finden. Zu jedem definierten regulären Ausdruck gibt es einen endlichen Automaten, der die vom regulären Ausdruck spezifizierte Sprache akzeptiert. Reguläre Ausdrücke können als Kriterium zum Filtern in der Textsuche verwendet werden.

Beispiel:

[1-9][ab] *steht für „*Eine Zahl 1-9 gefolgt von einem Buchstaben a oder b.“

[16] http://en.wikipedia.org/wiki/Regular_expression#POSIX_Basic_Regular_Expressions



```
base.txt | base.exp | base.txt
1  NONE            = '.*'                  : ''
2  NOTEMPTY        = '.*[^ ]+.*'           : 'should not be empty'
3  NOBLANK         = '[^ ]+'               : 'should not contain spaces'
4  ENOBLANK        = '()|(RE:NOBLANK)'     : 'should be empty or should not contain spaces'
5  NUMERIC         = '0|[1-9][0-9]*'       : 'should be numeric (decimal) without leading zero(s)'
6  ENUMERIC        = '()|(RE:NUMERIC)'     : 'should be numeric (decimal) without leading zero(s) or empty'
7  NUMERICLZ       = '[0-9]+'              : 'should be numeric (decimal) with or without leading zero(s)'
8  ENUMERICLZ      = '()|(RE:NUMERICLZ)'   : 'should be numeric (decimal) with or without leading zero(s) or empty'
9  DOT_NUMERIC     = '(RE:NUMERIC).[0-9]+'
10                 : 'should be numeric (decimal) with dot e.g. 5.0'
11 EDOT_NUMERIC    = '()|(RE:DOT_NUMERIC)'
12                 : 'should be numeric (decimal) with dot e.g. 5.0 or empty'
13 NUM_HEX         = '0x[[:xdigit:]]+'
14                 : 'should be a hexadecimal number (a number starting with "0x")'
15 ENUM_HEX        = '()|0x[[:xdigit:]]+'
16                 : 'should be empty or a hexadecimal number (a number starting with "0x")'
17 NUM_ANY         = '(RE:NUMERIC)|(RE:NUM_HEX)'
18                 : 'should be a decimal or hexadecimal number'
```

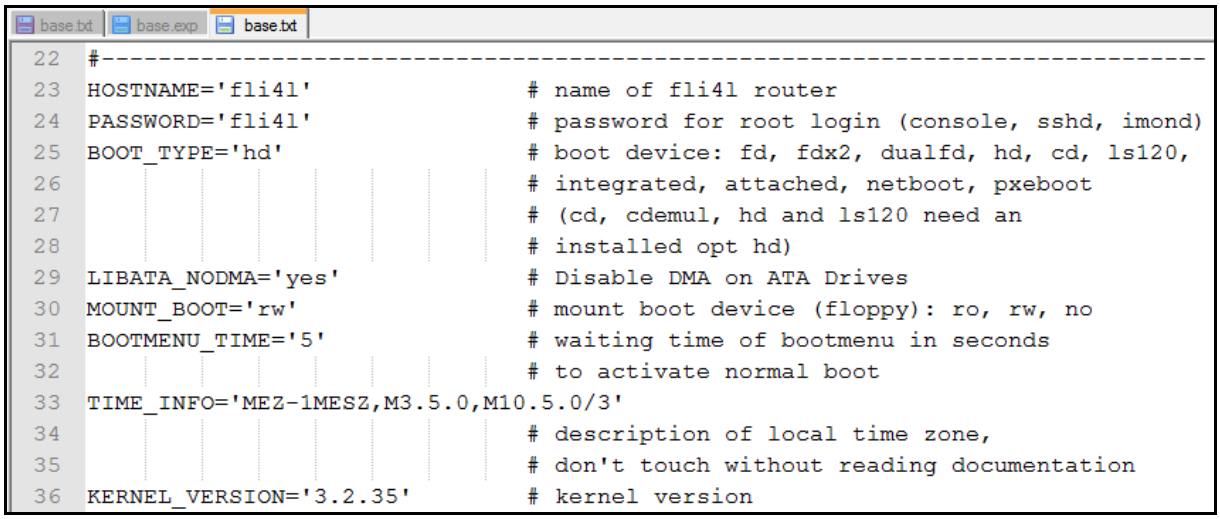
Abbildung 40: Auszug aus check/base.reg (Variablentyp)

5.4.3. Variablenzuweisung / Variable Assignment

In der Datei config/<Paketname>.txt werden die Variablen mit Werten gefüllt. Bei der Belegung von Variablen muss der Variablenname genannt und die Belegung in Hochkommata angeben werden. Um die Variable zu kommentieren können in derselben Zeile mit dem # - Zeichen Kommentare eingeleitet werden. Mit dem # - Symbol können auch ganze Variablenzeilen auskommentiert werden, damit diese nicht mehr beachtet werden. Leere Zeilen werden ebenfalls nicht beachtet.

Die Variablen werden dabei auf die regulären Ausrücke geprüft, mit denen die Variablentypen definiert wurden. Wurde eine Variable mit einem illegalen Wert gefüllt, führt dieses zu einer Fehlermeldung und zum Abbruch von MKFLI4L.

Exkurs MKFLI4L: Mit Hilfe von MKFLI4L werden in Abhängigkeit von der konkreten Konfiguration eine Datei rc.cfg erstellt, die für die Konfiguration zuständig ist, und zwei Archivdateien rootfs.img und opt.img, die alle Konfigurationsinformationen enthalten. MKFLI4L liest die einzelnen Pakete ein und prüft diese auf Fehler in der Konfiguration.



```
base.txt | base.exp | base.txt
22 #-------------------------------------------------------------------------------
23 HOSTNAME='fli4l'                # name of fli4l router
24 PASSWORD='fli4l'                # password for root login (console, sshd, imond)
25 BOOT_TYPE='hd'                  # boot device: fd, fdx2, dualfd, hd, cd, ls120,
26                                 # integrated, attached, netboot, pxeboot
27                                 # (cd, cdemul, hd and ls120 need an
28                                 # installed opt hd)
29 LIBATA_NODMA='yes'              # Disable DMA on ATA Drives
30 MOUNT_BOOT='rw'                 # mount boot device (floppy): ro, rw, no
31 BOOTMENU_TIME='5'               # waiting time of bootmenu in seconds
32                                 # to activate normal boot
33 TIME_INFO='MEZ-1MESZ,M3.5.0,M10.5.0/3'
34                                 # description of local time zone,
35                                 # don't touch without reading documentation
36 KERNEL_VERSION='3.2.35'         # kernel version
```

Abbildung 41: Auszug aus config/base.txt (Variablenzuweisung)

5.5. Fachlicher Entwurf

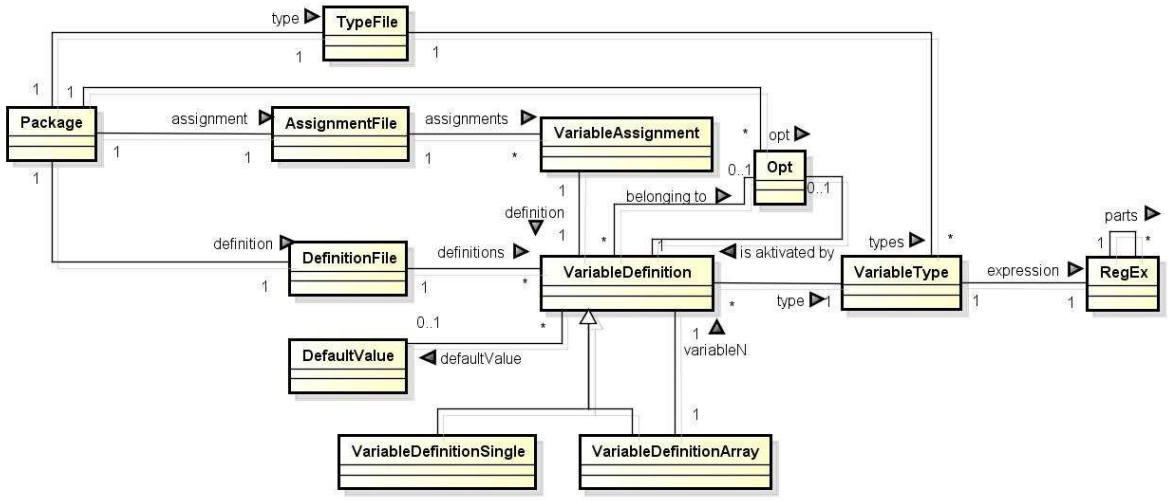


Abbildung 42: UML2 Klassendiagramm (fachlich)

Aus der vorangegangenen Analyse der Dateistruktur ergibt sich folgender Entwurf:

Ein „Package“ hat einen „TypeFile“, einen „AssignmentFile“, einen „DefinitionFile“ und beliebig viele „Opt“ Auf diese Weise bildet „Package“ die Paketstruktur von fli4l ab: Definition der Variablen, Typen und die Zuweisungen. Dabei besitzt jeder „TypeFile“ viele „VariableType“ (oder gar keine), die Einträge des TypeFile. Jeder „AssignmentFile“ hat viele „VariableAssignment“ (oder gar keine) und jeder „DefinitionFile“ unbegrenzt viele „VariableDefinition“. Jedes „VariableAssignment“ besitzt genau eine „VariableDefinition“, sprich jede Zuweisung braucht genau eine Variablendefinition. Die „VariableDefinition“ besitzt genau einen „VariableType“, hat null bis eins „Opt“ und null bis eins „DefaultValue“. Das soll heißen, dass jede Definition einer Variablen genaue einen Variablen-Typ braucht und eine oder keine optionale Variable hat und ebenfalls keinen oder einen Default-Wert. Die „DefaultValue“-Klasse, hat in der anderen Richtung unbegrenzt viele „VariableDefinition“ (der selbe „DefaultValue“ wird von mehreren „VariableDefinition“ genutzt). Die „VariableDefinition“ selber kann entweder eine „VariableDefinitionSingle“ oder eine „VariableDefinitionArray“ sein. Letzteres bildet Arrays ab. Ersteres steht für alle Definitionen bei denen kein Array angewendet wird. Die „VariableDefinitionArray“ hat immer genau eine „VariableDefinition“, sprich ein Array braucht immer eine Variablen-Definition zu der es gehört. Ein „VariableType“ hat genau eine Expression „RegEx“. Ein „RegEx“ wiederum hat unbegrenzt viele „parts“ (Teilausdrücke).

5.6. Technischer Entwurf

In der technischen Umsetzung bekommt die Klasse „Package“ das eindeutige String-Attribut „name“, damit man das einzelne Paket konkret ansprechen kann. Die Klasse „VariableAssignment“ enthält das Attribut „value“. „Value“ ist ein String-Attribut und beinhaltet den Wert, der einer Variablen zugewiesen wurde. Des Weiteren enthält diese Klasse die Operation „checkRegEx()“, die einen Boolean zurückgibt und die Variablen auf ihre Richtigkeit überprüft. Die Klasse „VariableAssignment“ hat, wie schon im Fachklassendiagramm beschrieben, genau eine Variablen-Definition und verweist deswegen auf die abstrakte Klasse „VariableDefinition“. Diese enthält die Attribute „name“ und „isOptional“. Ersteres ist ein String-Attribut und gibt der Variablen einen Namen, der eindeutig ist und hilft

diese konkret ansprechen zu können. Letzteres ist ein boolean-Attribut und gibt an ob eine Variable vorkommen muss oder ob diese nur optional genutzt werden kann.

Um in der Klasse „VariableDefinition“ null bis eins optVariablen umzusetzen zu können, gibt es eine Referenz auf das Interface „VariableReference“, dieses kann entweder durch eine Klasse „HasOpt“ oder „HasNotOpt“ repräsentiert werden. Jedes „HasOpt“ hat ein „Opt“. Dieses hat das Attribut „name“ und zeigt auf die „VariableDefinition“ die für das OPT steht. „Name“ ist wie bisher ein eindeutiges Attribut, mit dem das „Opt“ konkret angesprochen werden kann. „VariableDefinition“ zeigt auf das Interface „DefaultValue“, dieses kann entweder durch die Klasse „HasDefault“ oder „HasNotDefault“ repräsentiert werden. Ersteres hat ein String-Attribut „defaultValue“, das den Standardwert repräsentiert. Letzteres bedeutet, es gibt keinen Default-Wert.

Wie schon im Fachklassendiagramm hat jede „VariableDefinition“ genau einen „VariableType“. Die Klasse „VariableType“ hat zwei String-Attribute: „name“ und „errorMessage“. „Name“ ist auch hier eindeutig und zum Ansprechen der einzelnen Variablen Typen da. Die „errorMessage“ dient dem Fehlertext, der in der Datei hinter dem Doppelpunkt angegeben wird. Um die regulären Ausdrücke darzustellen, hat die Klasse „VariableType“ eine expression (Ausdruck) „RegEx“. „RegEx“ hat die Operation „getStringRepresentation“. Diese Operation gibt in der Variablen „rep“ den zusammengesetzten Wert zurück, der aus den Werten der Liste von „parts“ besteht. Diese Werte werden von der Operation „getMyStringPart“ geliefert. „GetMyStringPart“ kommt in den Klassen „RegExPart“ und „RefState“ vor. Die Liste der regulären Ausdrücke ist geordnet, also die Inhalte werden in derselben Reihenfolge wieder ausgegeben, wie sie zuvor eingefügt wurden. Das Interface „RegExPart“ kann durch „RegExRefPart“ oder „RegExStringPart“ repräsentiert werden. „RegExStringPart“ enthält die regulären Ausdrücke. Diese werden im Attribut „value“ gespeichert. „RegExRefPart“ enthält Referenzen auf regulären Ausdrücken und die Operation „resolve“. Jeder „RegExRefPart“ hat genau ein „RefState“. Die Operation „resolve“ ändert den Status eines „RefUnResState“ in „RefResState“. „RefState“ ist eine abstrakte Klasse und wird durch „RefUnResState“ und „RefResState“ repräsentiert. Damit die regulären Ausdrücke in „RefUnResState“ aufgelöst werden können, hat jeder „RefUnResState“ genau einen „VariableType“.

Zum Erstellen der regulären Ausdrücke wurde die Java-Bibliothek brics automaton verwendet. Brics automaton ist eine sehr schlanke Bibliothek und unterstützt die Standard regulären Ausdrücke (Kontamination, Vereinigung, Kleensche Hülle) und einige zusätzliche Ausdrücke, wie beispielsweise Überschneidungen. Wichtig für die Auswahl der RegEx-Bibliothek war, dass sie nicht zu komplex ist. Dieses war zum Beispiel bei der Java-Standardbibliothek der Fall. Zu viel würde bedeuten, dass man auch Ausdrücke eingeben könnte, die zwar von unserem Modell, aber nicht von fli4l erkannt werden würden. Nachteil der Brics automaton Bibliothek ist, dass es keine Zeichenklassen unterstützt, wie beispielsweise ***[:space:]*** für alle Whitespace: Horizontaler und vertikaler Tabulator, Zeilen- und Seitenvorschub, Wagenrücklauf und Leerzeichen. Dieses Problem muss im Laufe des Integrationsprojekt noch behoben werden, da es in den zukünftigen Arbeiten zu Problemen führen wird. Weitere Information sowie die Dokumentation von brics automaton findet man auf der Webseite http://www.brics.dk/automaton/.

Für folgende Ausdrücke muss also noch für Unterstützung gesorgt werden:

Character Class	ASCII	Description
[:alnum:]	[A-Za-z0-9]	Alphanumeric characters

[:alpha:]	[A-Za-z]	Alphabetic characters
[:blank:]	[\t]	Space and tab
[:digit:]	[0-9]	Digits
[:graph:]	[\x21-\x7E]	Visible characters
[:lower:]	[a-z]	Lowercase letters
[:print:]	[\x20-\x7E]	Visible characters and the space character
[:punct:]	[\]\[!"#$%&'()*+,./:;<=>?@\^_`{|}~-]	Punctuation characters
[:space:]	[\t\r\n\v\f]	Whitespace characters
[:upper:]	[A-Z]	Uppercase letters
[:xdigit:]	[A-Fa-f0-9]	Hexadecimal digits

Tabelle 15: Standard-Zeichenklassen

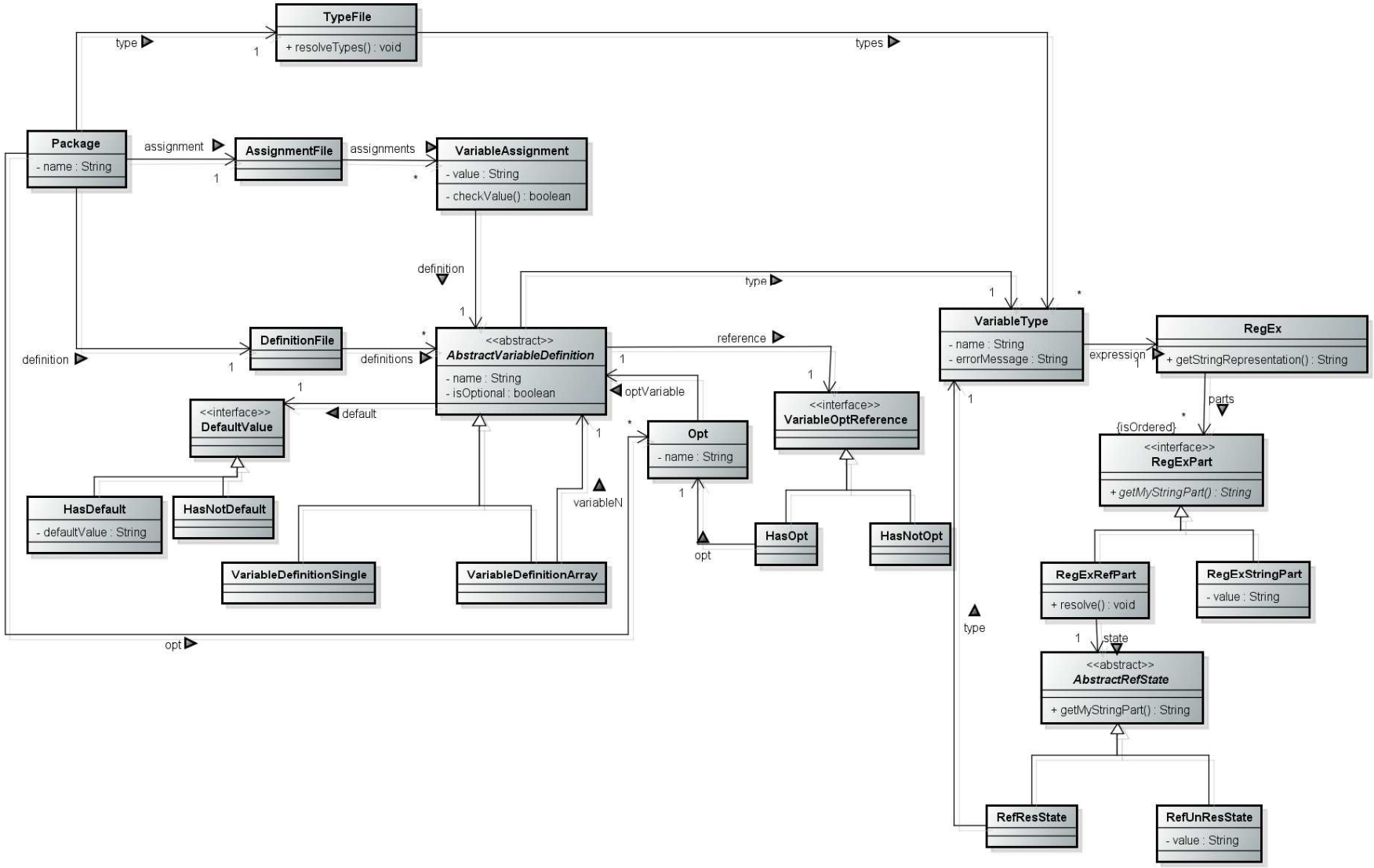


Abbildung 43: UML2-Klassendiagramm (technischer Entwurf)

5.7. Eclipse/Java- Projektstruktur

In der eclipse-Struktur gibt es zwei Ordner mit Quelltext, die jeweils gleich aufgebaut sind. Durch die identische Package-Struktur des „src“ und des „test“ Ordners können auch Methoden, die den Sichtbarkeitsmodifikator „package“ haben, direkt getestet werden. „Src“ teilt sich auf in einen „basic“-Teil für die Konstanten und einen „model“-Teil für die Modellwelt. Diese ist so implementiert, wie es das Entwurfsdiagramm zeigt. Die packages im „test“-Ordner sind wie bereits erwähnt identisch aufgebaut, zusätzlich gibt es eine Klasse „PackagePrepare“ in der eine Objektwelt zum Testen generiert wird. Für die *-Beziehungen wurde größtenteils java.util.HashMap verwendet und als „Key“ der jeweilige Zielname benutzt (dieser ist auch im Modell einzigartig).

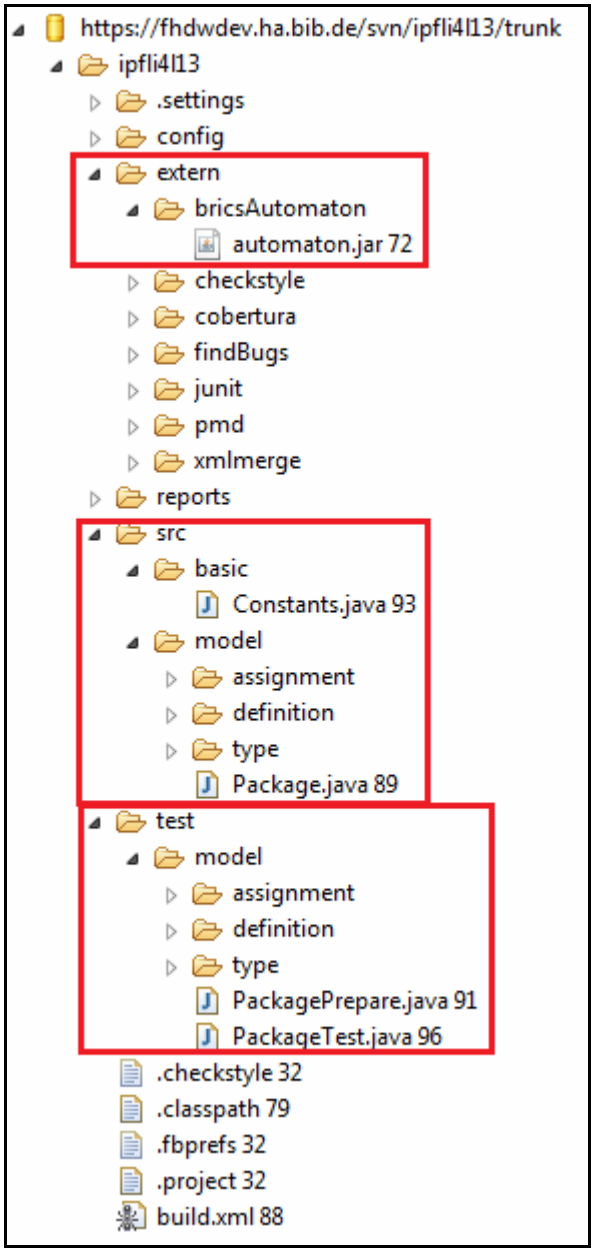


Abbildung 44: Projektstruktur

6. Scanner

6.1. Einführung

Ziel der Verarbeitung der Konfigurationen ist das Einlesen und Schreiben im Fli4l-Format. Hierzu wird zuerst ein Scanner benötigt, welcher die verschiedenen Dateien der Fli4l-Konfiguration einliest. Da die Daten folglich an den Parser weitergegeben werden sollen, müssen die Ergebnisse des Scanners in einer für den Parser verständliche Form generiert werden.
Der Parser analysiert die Informationen, welche er vom Scanner erhalten hat und hält diese im bereits entwickelten Modell fest.
Somit ist eine akkurate Kommunikation zwischen dem Scanner und dem Parser ein wichtiger Bestandteil zu einer erfolgreichen Entwicklung. Zuletzt ist das Drucken durch einen Printer vorgesehen.

6.2. Grundlagen

Die Grundvoraussetzung für das Einlesen der Dateien ist, dass der Scanner bereits einen Datenstrom mit dem Inhalt der Datei erhalten hat, den er einlesen (engl.: scannen) soll. Der Zugriff in das Dateisystem ist somit nicht bei der Entwicklung des Scanners zu berücksichtigen.
Grundsätzlich gibt es drei Dateitypen, welche vom Scanner eingelesen werden sollen. Somit werden drei Scanner benötigt, wobei jeder Scanner für genau einen Dateityp verantwortlich ist. Diese Entscheidung wurde aufgrund der Unterschiede, der in den Dateien verwendeten Zeichen, getroffen.

Zum einen werden die Dateien eingelesen, welche keine Abhängigkeit zu den anderen Dateien aufweisen, die „Typdateien". Diese enthalten die grundlegenden Typen des Fli4l-Projekts. Der Dateiname ist dabei unerheblich wichtig ist lediglich die Endung „.exp". Das Scannen dieser Typdateien führt der so genannte „Typefilescanner" durch.

Weiterhin sind „Definitionsdateien" vorhanden, die Variablen definieren. Die Definitionsdateien werden im Projekt durch den „Definitionfilescanner" eingelesen. Die Definitionen werden in Textdateien (.txt) angegeben.

Des Weiteren werden die Zuweisungsdateien gescannt, welche im Projekt „Assignmentfile" genannt werden. Diese sind, wie die Definitionsdateien, im Textformat vorhanden und werden durch den „Assignmentfilescanner" eingelesen.

Alle drei Scanner besitzen einen eigenen Puffer, in dem die Scanner die eingelesenen Symbole für den nachfolgenden Parser bereitstellen und somit als Pipeline zwischen den beiden Komponenten, Scanner und Parser, dient.

Abbildung 45: Zusammenhang Scanner, Parser und Printer

6.3. Analyse

Die folgenden Anwendungsfälle wurden anhand der Aufgabenstellung erstellt, anhand derer die Analyse durchgeführt werden konnte.

6.3.1. Anwendungsfälle

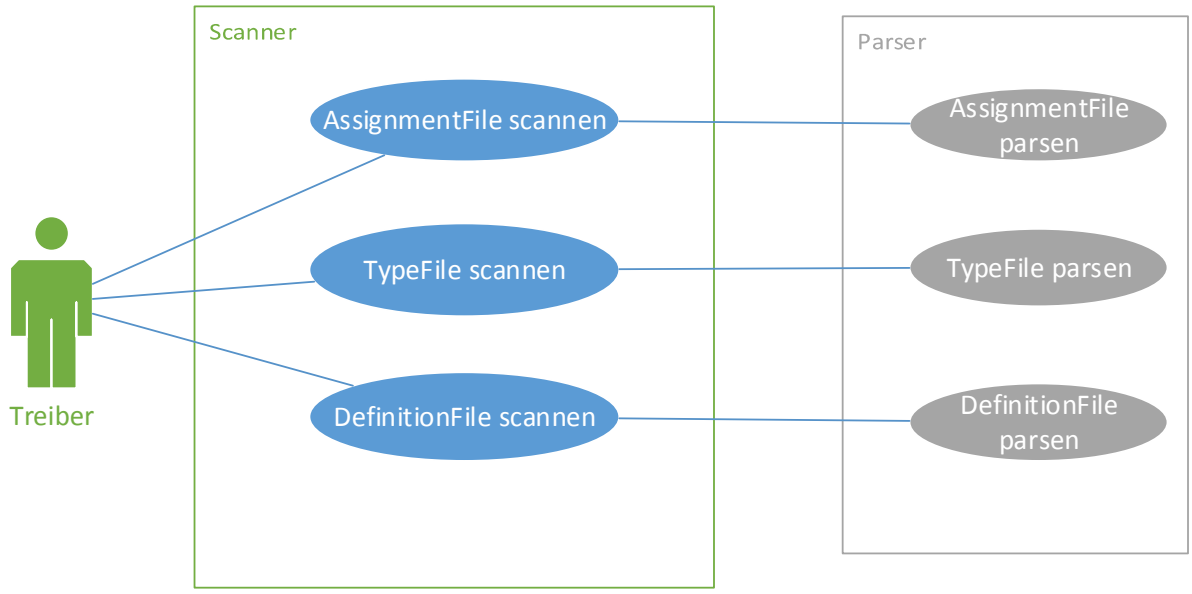


Abbildung 46 - Anwendungsfalldiagramm

Das Anwendungsfalldiagramm in Abbildung 2 zeigt die Vorgänge, welche durch einen so genannten „Treiber“ hervorgerufen werden. Der Treiber ist für die Steuerung des Scanners verantwortlich. Der Treibersollte die Fähigkeit besitzen, das nebenläufige Ausführen von Scanner, Parser und Printer durch die Erstellung und Zuweisung von Threads zu ermöglichen. Der Treiber gibt den Befehl des Scannens durch den Aufruf der *toSymbolSequence*-Methode mit der einzulesenden Datei als Parameter. Da die Scannerwahl von der Eingangsdatei abhängig ist, muss einer Datei durch den Treiber ein spezieller Scanner zugeordnet und gegebenenfalls instanziiert werden. Der Dateiname wird über den Scanner an den Parser weitergeben.

6.4. Entwurf

Anhand der Analyse und der Anwendungsfälle, konnte ein konkreter Entwurf der späteren Implementierung erstellt werden.

6.4.1. Allgemeine Funktionsweise der Scanner

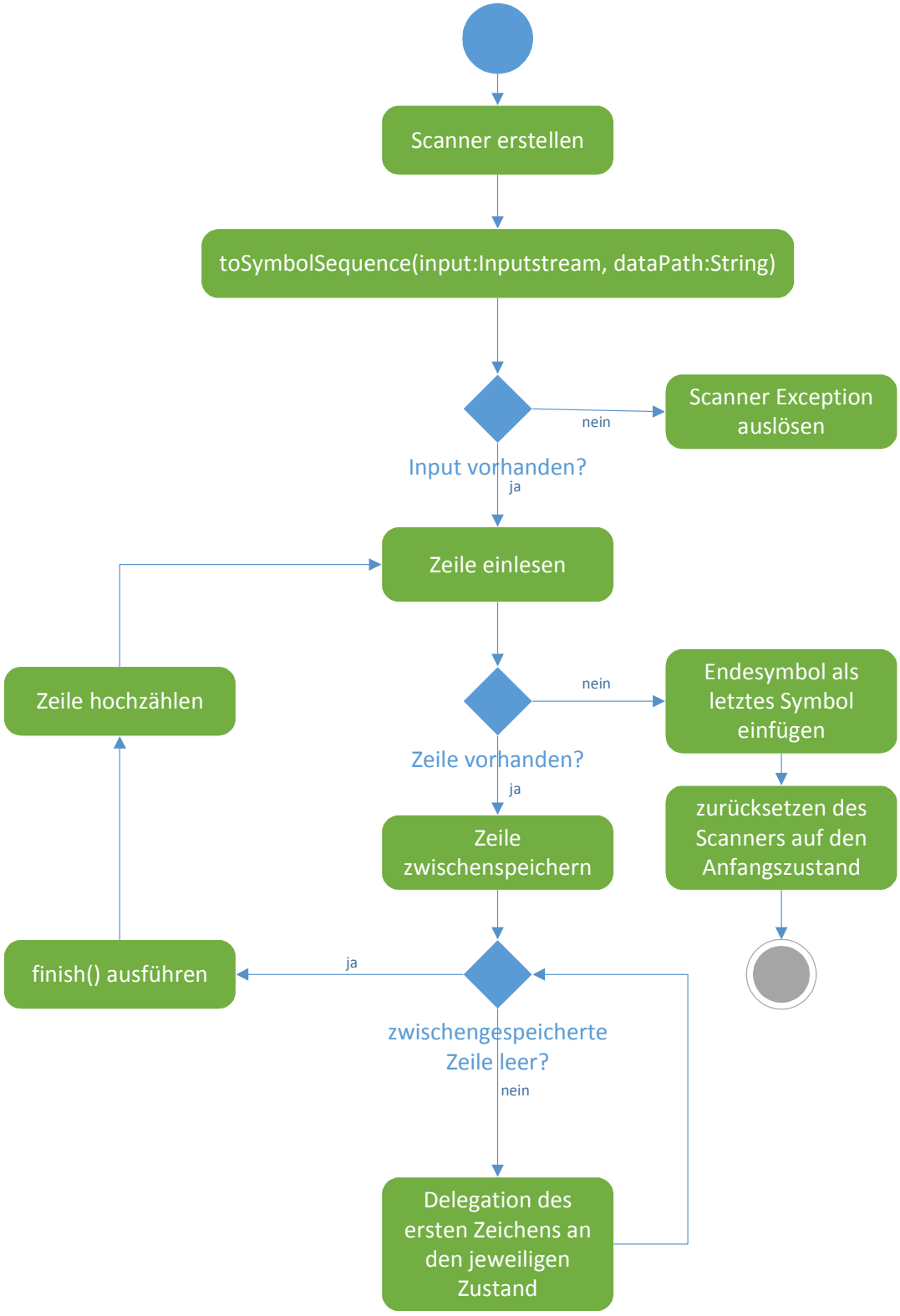


Abbildung 47 - Funktionsweise des Scanners

Wie in Abbildung 3 zusehen, wird gegebenenfalls zuerst ein Scanner instanziiert und anschließend die *toSymbolSequence*-Methode aufgerufen. Als Eingabeparamter wird ein aus der Eingabedatei erzeugter Eingabestrom und die Angabe des Dateinamens mit dem übergeordneten Ordner erwartet. Bei der Angabe des Dateinamens ist die Syntax „package/name.fileformat“ zu beachten. Im An-

schluss wird die Eingangsdatei durch den jeweiligen Scanner zeilenweise eingelesen und verarbeitet. Wurde eine Zeile eingelesen, wird diese in dem so genannten „momentanen Ausdruck“ (engl.: *current expression*) gespeichert. Die Methode finish() forciert das Erstellen eines Objektes auch aus den letzten Zeichen einer Zeile, indem die bereits gesammelten, aber noch nicht zu einem Objekt verarbeiteten Zeichen in ein Symbolobjekt transformiert werden und der Auswahlzustand als aktueller Zustand gesetzt wird.
Solange der momentane Ausdruck noch Zeichen beinhaltet, wird jedes Zeichen eingelesen und einem jeweiligen Zustand zugewiesen, der dieses Zeichen bearbeitet. Dies wird von der Methode *state.action()* durchgeführt. Zuvor wird außerdem die finish()-Methode des aktuellen Zustands aufgerufen, der Zeilenzähler inkrementiert und der Spaltenzähler auf den Anfangszustand zurückgesetzt.
Wenn die Eingangsdatei keine weiteren Zeilen beinhaltet, wird dem Puffer ein so genanntes Endesymbol (*endSymbol)* hinzugefügt, welches dem Parser signalisiert, dass die Eingangsdatei komplett gescannt wurde.

6.4.2. Klassendiagramme

Scanner

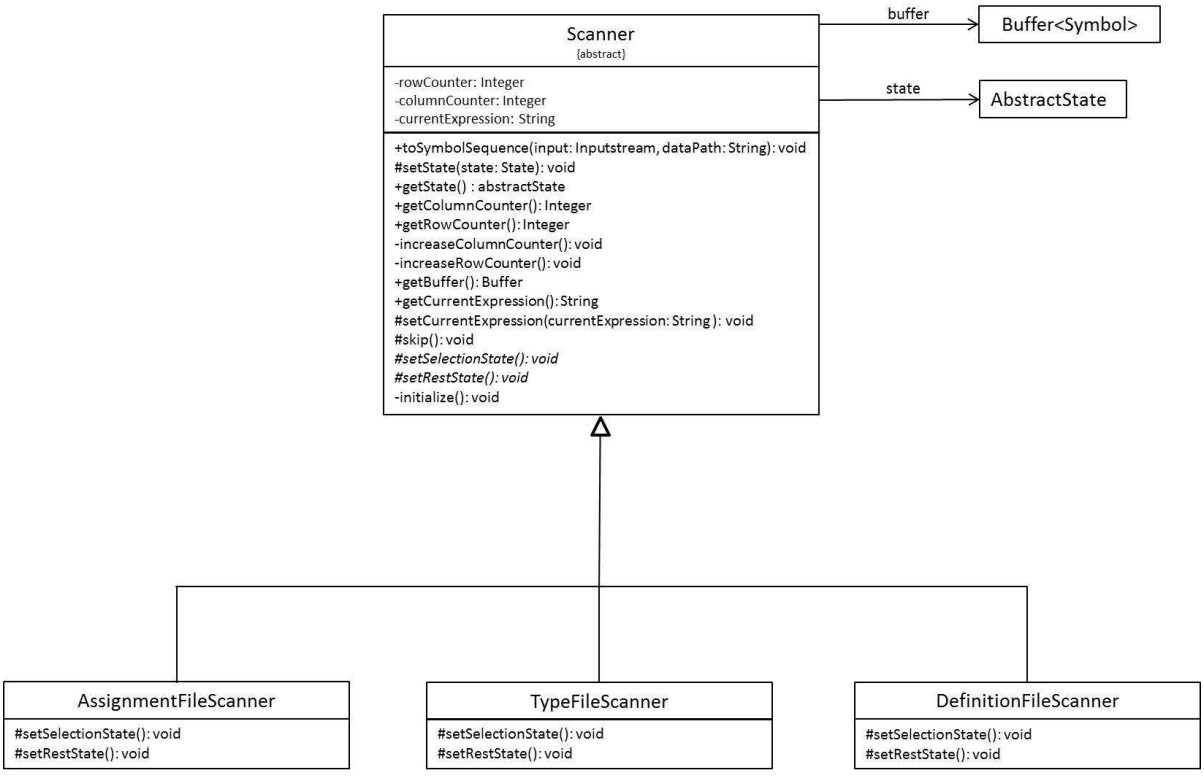


Abbildung 48 - Klassendiagramm Scanner

Das Klassenmodell verdeutlicht, dass das Scannen auf drei Scanner verteilt wurde. Für jeden Typ von Eingangsdatei, ist ein bestimmter Scanner mit einem eigenen Puffer vorhanden. Dieser Puffer dient dazu dem Parser die gescannten Symbole zur Entnahme bereit zu stellen. Des Weiteren besitzt jeder Scanner zu jedem Zeitpunkt einen Zustand und hat Zähler, welche die Zeilen- und Spaltennummern vorhalten. Jede vom Scanner zu bearbeitende Zeile wird in einem momentanen Ausdruck zwischengespeichert. Diese Zwischenspeicherung wird mittels des Zeichenkettenattributs *currentExpression* getätigt.

Jeder Scanner besitzt zusätzlich zu den bereits erläuterten Attributen Methoden. Die Methode *toSymbolSequence(…)* ermöglicht die Erstellung von Symbolsequenzen aus einem zu scannenden Ausdruck. Die Methoden *increaseRowCounter()* und *increaseColumnCounter()* haben lediglich die Aufgabe, die bereits erwähnten Zeilen- beziehungsweise Spaltenzähler zu inkrementieren, um die Positionen der Symbole zu dokumentieren. Die Methode *skip()* löscht das erste Zeichen des momentanen Ausdrucks und inkrementiert den Spaltenzähler. Die Methode *initialize()* wird benötigt, um am Ende einer Datei den Scanner wieder auf den Anfangszustand zurückzusetzen. Das heißt, dass die Zähler auf den Anfangszustand zurückgesetzt und der momentane Ausdruck geleert wird.

Zustand

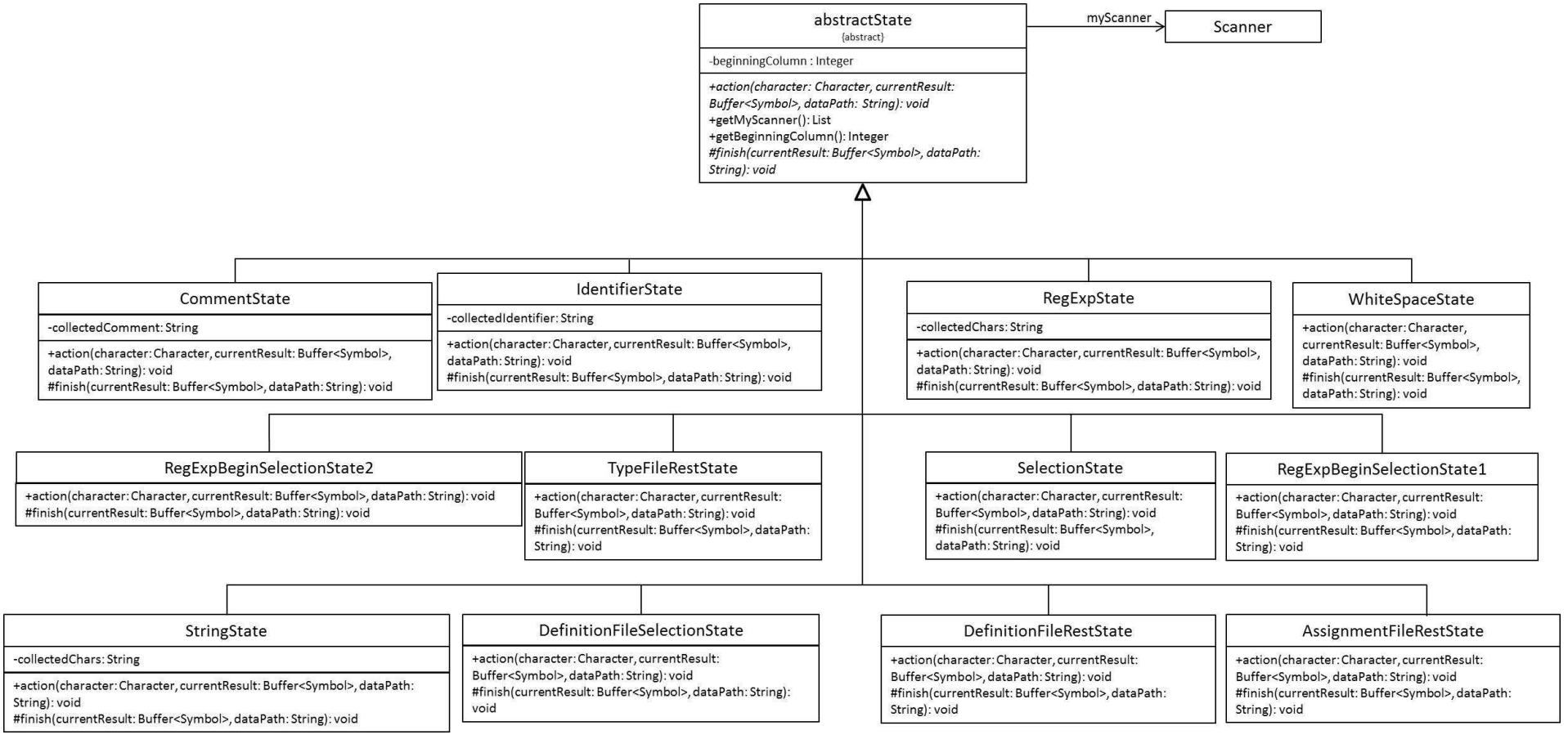


Abbildung 49 - Klassendiagramm Zustände

Dieses Klassenmodell zeigt die verschiedenen Zustände, die während des Einlesens der Eingangsdatei eingenommen werden können. Der momentane Zustand des Scanners ist abhängig von den eingelesenen Zeichen des momentanen Ausdrucks. Nochmals hervorzuheben ist hier der Auswahlzustand, welcher das Delegieren an die anderen Zustände ermöglicht. Ist eine Zeile fertig gescannt, muss der aktuelle Zustand der Auswahlzustand sein. Die Wichtigkeit dieses Sachverhalts ist bei einem Kommentar am ehesten zu verdeutlichen. Ein Kommentar wird erst durch einen Zeilenumbruch beendet. Würde nach dem Kommentar der Scanner nicht im Auswahlzustand sein, wäre der letzte Zustand des Scanners beim Einlesen der Zeile der Kommentarzustand. Da erst beim Übergang in den Auswahlzustand ein Symbolobjekt erzeugt wird, würde das Erstellen des zugehörigen Objektes fehlen, wenn kein Übergang zum Auswahlzustand stattfindet. Durch diesen letzten Übergang wird sichergestellt, dass auch von diesem letzten Kommentar ein Objekt erzeugt wird. Die Methode finish() ermöglicht diese letzte Transformationen von Zeichen in Symbole.

Die abstrakte Klasse *AbstractState* beinhaltet mehrere Attribute, welche im Folgenden erläutert werden. Durch das Attribut *myScanner*, wird jedem Zustand ein Scanner zugeordnet. Weiterhin muss ein Verweis auf das erste Zeichen eines Symbols vorhanden sein. Es genügt hierbei jedoch die Spalte, da sich ein Symbol nicht über mehrere Zeilen erstrecken darf. Dies wird hier durch das Attribut *beginColumn dokumentiert*. Neben den Attributen sind Methoden vorhanden, welche von den verschiedenen Zuständen spezialisiert werden. Die abstrakte Methode action() muss von jedem State implementiert werden.

Manche Zustände, wie der Zustand der regulären Ausdrücke (*RegularExpressionState*), besitzen neben ihren Konstruktoren und der Aktionsmethode ein Attribut namens Wert, Ausdruck oder Text, je nach Zustandstyp. Diese Attribute dienen zur Sammlung von Zeichen, falls der Scanner beim Scannen mehrerer Zeichen in einem Zustand verbleibt. Die jeweiligen Attribute dienen dann als eine Art Sammelstelle für diese Zeichen.

Symbole

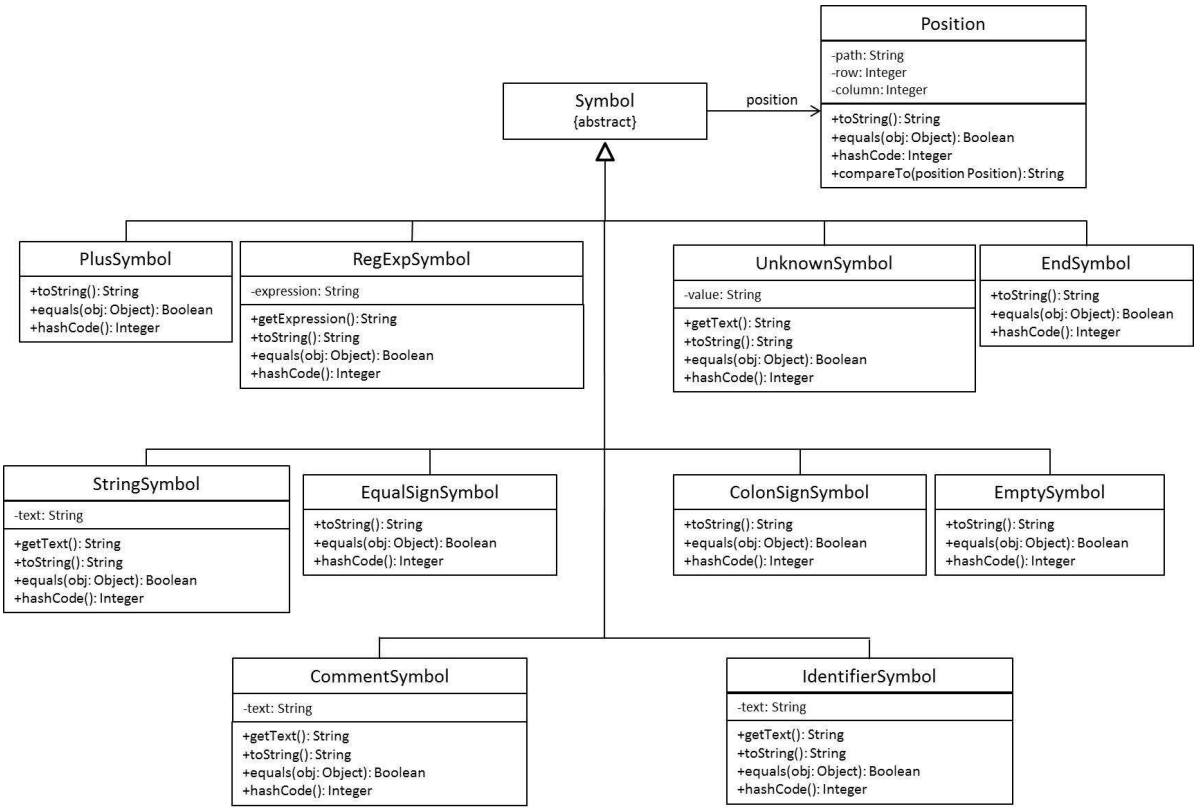


Abbildung 50 - Klassendiagramm Symbole

Dieses Klassenmodell zeigt die verschiedenen Symbole, in welchen die eingelesenen Zeichen transformiert werden. Hervorzuheben ist hier das Endesymbol, welches an die letzte Stelle des Puffers hinzugefügt wird, um dem Parser zu signalisieren, dass die Datei fertig gescannt wurde. Jedes Symbol besitzt dabei eine Positionsangabe, welche durch das Attribut *position* dargestellt wird. Die Positionsangabe gibt an, in welcher Spalte und Zeile welches Dateipfades das erste Zeichen eines Symbols in der Eingabe erkannt wurde. Hierbei ist wichtig, dass bei einem Kommentarsymbol und einem Zeichenkettensymbol die Zeichen # bzw. “ oder ‘ als erstes Zeichen zählen. Außerdem ist wichtig, dass die Zeile und die Spalte der Position in der aktuellen Einstellung bei 0 beginnen.

6.4.3. Zustände

Typdateiscanner

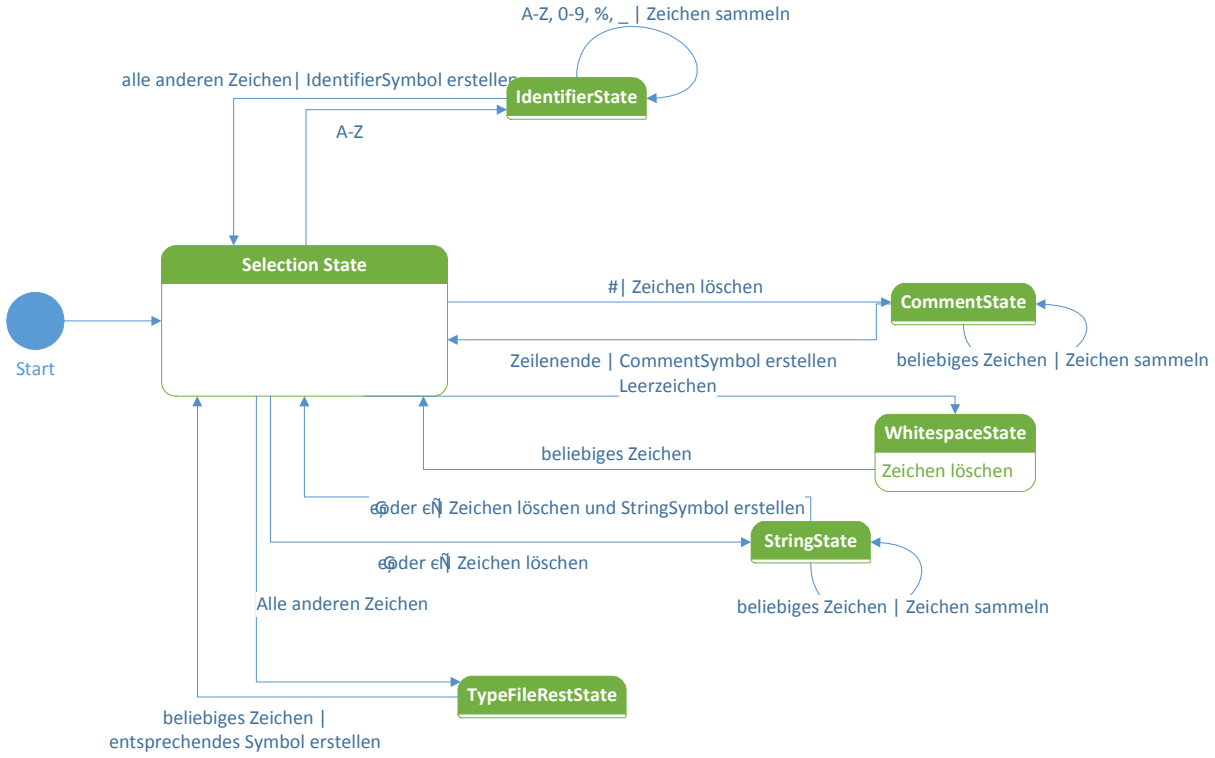


Abbildung 51 - Zustände Dateitypscanner

In dieser Abbildung ist zu erkennen, dass das State-Pattern genutzt wird, um die Scannertypen zu realisieren. Nach dem Einlesen einer Zeile ist der Auswahlzustand *SelectionState* der aktuelle Zustand, welcher an die weiteren Zuständen delegiert, aber auch von diesen wieder erreicht wird.
Die verarbeitenden Zustände werden durch verschiedene Zeichen vom Scanner eingenommen, die er einliest. Die Zeichen werden in Kommentar-, Bezeichner-, Zeichenketten-, Leerzeichen- und Restsymbole transformiert.
Die Zeichen, die zum Übergang in einen Zustand nötig sind, sind jeweils an den Pfeilen vermerkt, welche vom Auswahlzustand weg führen. Zeichen, welche das Verlassen des Zustands forcieren, sind an den zum Auswahlzustand hin führenden Pfeilen erkennbar. Aktionen eines Zustands werden hinter einem „|“ genannt. Wird ein verarbeitender Zustand wieder verlassen, wird ein Symbol aus den eingelesenen Zeichen erzeugt. Zustände, in denen verblieben wird sammeln die eingelesenen Zeichen, solange der jeweilige Zustand nicht verlassen wird.
Aus der Abbildung geht ebenfalls hervor, dass mehrere Zeichen in einen Zustand führen können. So kann ‘ oder “ eine Zeichenkette einleiten oder alle Buchstaben von A bis Z für den Anfang einer Bezeichnung stehen. Im Folgenden werden die Zustände im Einzelnen erläutert.
Der *Restzustand* verarbeitet alle Zeichen, die den anderen Zuständen nicht zugeordnet werden können. Dabei unterscheidet der Restzustand zwischen bekannten und unbekannten Zeichen. Hinzuzufügen ist, dass der Restzustand aus unbekannten und bekannten Zeichen Symbole erstellt.
Die Zusammenfassung von bekannten und unbekannten Symbolen in einem Zustand geschieht, da der Typdateiscanner sowie der Zuweisungsdateiscanner denselben Auswahlzustand verwenden können, auch wenn diese Restzustände verschiedene Zeichen unterschiedlich behandeln.

Um die Performanz des Restzustands zu erhöhen, wäre es möglich gewesen, unbekannte Zeichen zu sammeln, um diese gemeinsam als unbekanntes Symbol zu deklarieren. Somit könnte verhindert werden, dass bei zwei aufeinanderfolgenden Zeichen, welche beide durch den Restzustand verarbeitet werden, ein Zustandswechsel vom Restzustand in den Auswahlzustand und dann wieder in den Restzustand geschieht. Diese Vorgehensweise wurde nicht implementiert, da dadurch eine Abhängigkeit des Restzustands vom nächsten Zeichen entstehen würde. Die Abhängigkeit entstünde durch den Verbleib des Scanners im Restzustand, solange kein bekanntes Zeichen eingelesen wird. Um dieses Verhalten zu implementieren, wäre eine weitere Abfrage nötig, welche die Funktionsweise des Auswahlzustands imitieren würde.
Die Erweiterbarkeit und Flexibilität des Scanners wären eingeschränkt, da beim Hinzufügen von mehrstelligen Zuständen die Änderungen an mehreren Stellen getätigt werden müssten. Ein mehrstelliger Zustand ist beispielsweise der Bezeichnerzustand, da dessen Symbole aus mehreren Zeichen bestehen können. Eine Coderedundanz wäre somit unumgänglich.
Besonders für den Restzustand des Typdateiscanners ist, dass ein Doppelpunkt als ColonSignSymbol und ein Gleichheitszeichen als EqualSignSymbol bekannte Zeichen sind. Hingegen ein Bindestrich kommt nicht in der Datei vor und stellt somit für den Scanner ein unbekanntes Zeichen dar. Die Unterscheidung zwischen bekannten- und unbekannten Zeichen findet durch die Restzustände der jeweiligen speziellen Scanner statt.
Der *Leerzeichenzustand* sticht in seiner Aufgabe hervor, da dieser nicht wie die anderen Zustände aus den Zeichen Symbolobjekte erzeugt, sondern die eingelesenen Leerzeichen löscht. Somit werden aus Leerzeichen keine Symbole erzeugt, außer sie befinden sich innerhalb einer Zeichenkette. Dies ist der Grund, warum der Leerzeichenzustand vom Restzustand getrennt wurde, da der Restzustand immer Symbole erzeugt, sogar wenn die eingelesenen Zeichen unbekannt sind.
Der *Zeichenkettenzustand* erstellt Zeichenketten, wodurch die Möglichkeit des Sammelns mehrerer Zeichen eine große Bedeutung hat. Die Zeichenketten werden dabei ohne die Einleitenden- beziehungsweise abschließenden Anführungsstriche an den Parser weitergegeben.
Der *Bezeichnerzustand* besitzt lediglich die Aufgabe Bezeichnungen zu erkennen und diese im Puffer zu speichern. Im Gegensatz zu den Zeichenketten werden die Bezeichnungen nicht mit Anführungsstrichen eingeleitet beziehungsweise abgeschlossen.
Der *Kommentarzustand* ermöglicht das Verarbeiten von Kommentaren, welche durch ein „#“ eingeleitet werden. Diese Ende durch einen Zeilenumbruch und werden anschließend im Puffer gespeichert.

Definitionen

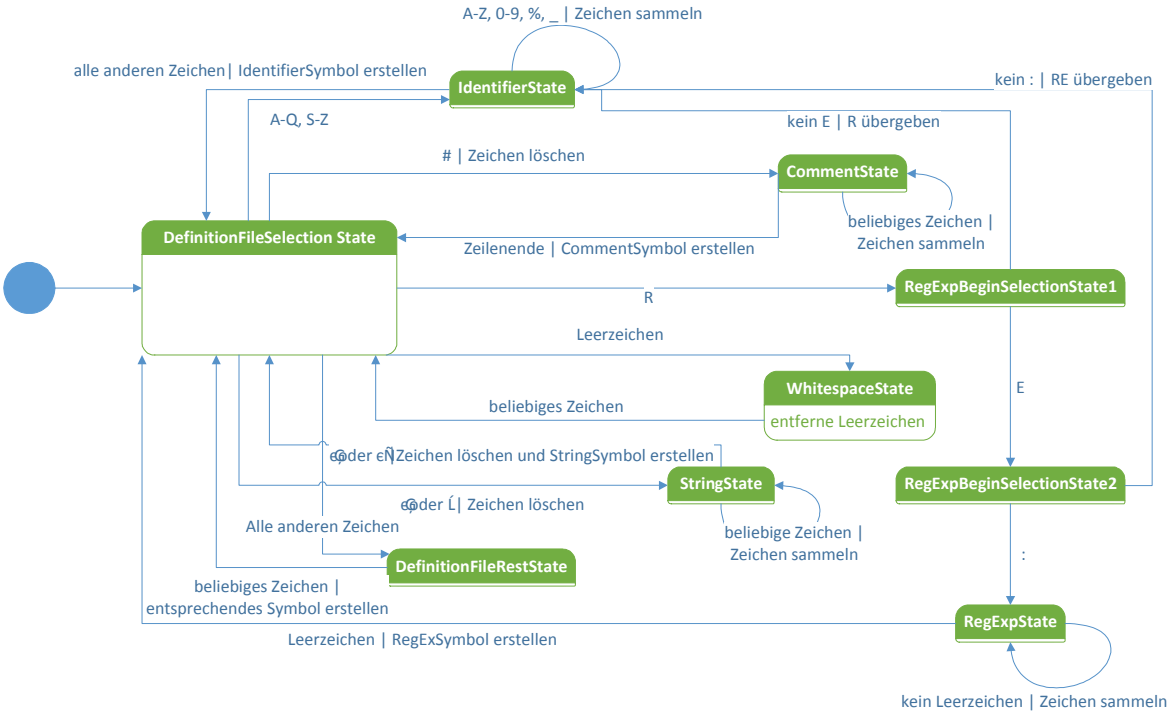


Abbildung 52: Zustände Definitionsdateiscanner

Die grundsätzliche Funktionsweise des Definitiondateiscanners ist der des Typdateiscanners sehr ähnlich. Auch hier werden durch Zustandswechsel die eingelesenen Zeichen in bestimmte Symbole transformiert. Die Funktionsweisen der meisten Zustände sind dem des Typdateiscanners gleich, bis auf den Restzustand und der Zustände der regulären Ausdrücke, welche im Typdateiscanner anders beziehungsweise nicht vorhanden sind.

Die Besonderheit der Definitionsdatei bezüglich ihrer Symbolvielfalt besteht in der zusätzlichen Einbindung von regulären Ausdrücken. Diese werden durch „RE:“ eingeleitet, also durch drei hintereinander folgenden Zeichen. Wird ein „R“ zu Beginn eingelesen, wird das „R“ aus dem Ausdruck entfernt und der Scanner wechselt seinen Zustand in den „RegExpBeginSelectionState1“. Folgt darauf ein „E“ wird der RegExpBeginSelectionState2 erreicht. Folglich fehlt nur noch das Zeichen „:“ und ein regulärer Ausdruck wurde erkannt. Nun werden alle darauf folgenden Zeichen bis zum nächsten Leerzeichen als regulärer Ausdruck verarbeitet. Ein Leerzeichen führt zur Beendigung der Sammlung der Zeichen. Das Leerzeichen wird zunächst an den Auswahlzustand weitergegeben und es wird ein RegExpSymbol erstellt. Dadurch wird der Übergang des Scanners in den Der Abhängigkeit zwischen dem Bezeichnerzustand und den Zuständen der regulären Ausdrücke ist hier von besonderer Bedeutung. Wird während eines Bezeichnerzustands ein R gefunden, so kann dieses Zeichen keinen regulären Ausdruck einleiten, sondern gehört zu der Bezeichnung.

Wenn nach dem ersten oder zweiten regulärem Ausdruckszustand ein Zeichen eingelesen wird, welches nicht den Zustände der regulären Ausdrücke zuordnungsbar ist, führt dieses in den Bezeichnerzustand um ein Zeichen, welches dem Bezeichnerstatus zugehörig ist, zu verarbeiten. Der Restzustand des Definitionsdateiscanners unterscheidet sich vom Restzustand des Typdateiscanners, da der Restzustand des Definitionsdateiscanners den Doppelpunkt nicht erkennt.

Zuweisungsdateiscanner

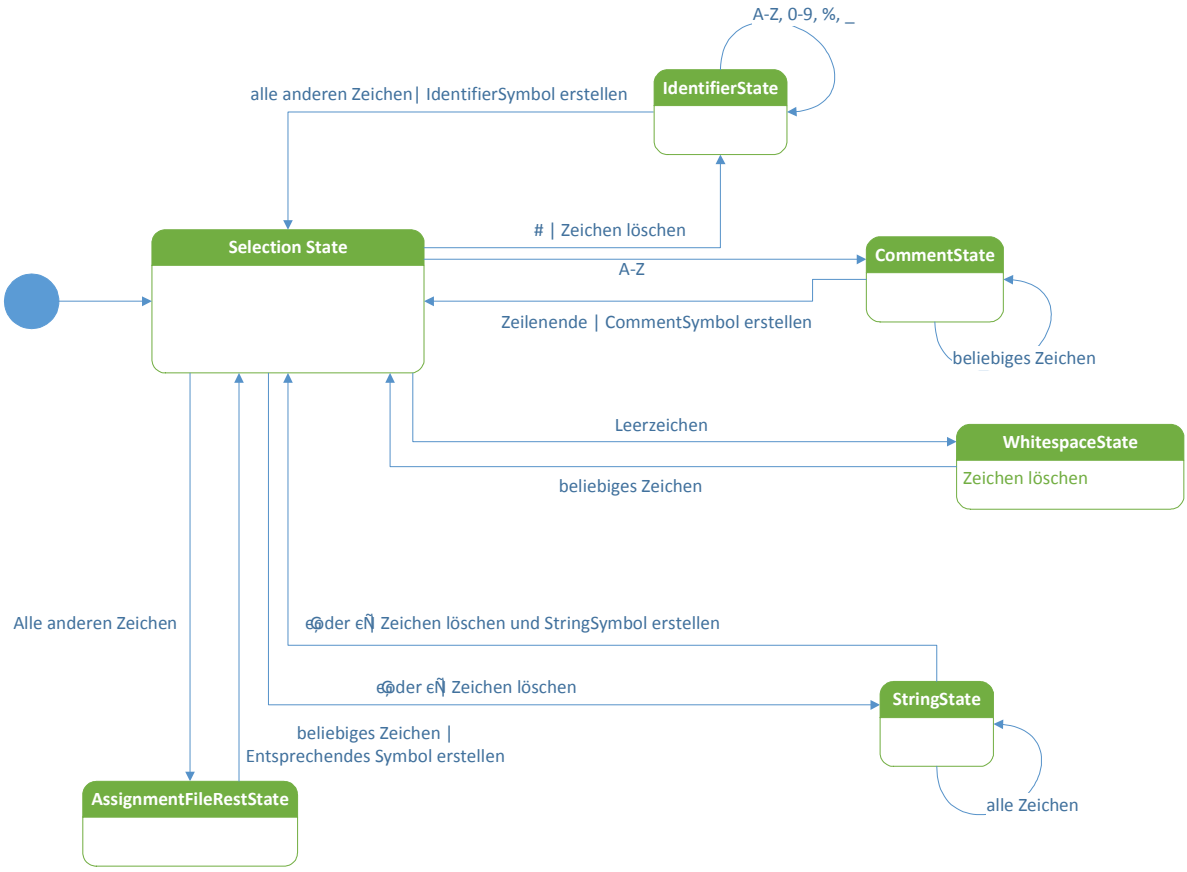


Abbildung 53 - Zustände Zuweisungsdateiscanner

Der Zuweisungsdateiscanner beinhaltet keine weiteren besonderen Eigenschaften. Da hier keine Referenzen vorhanden sind, ist die Ähnlichkeit zum Typdateiscanner sehr groß. Lediglich die Restzustände unterscheiden sich. Im Zuweisungsrestzustand werden Doppelpunkte, Bindestriche und Pluszeichen nicht zu bekannten Symbolen transformiert.

6.5. Umsetzung

Die Scanner, Symbole, und Zustände sind, wie im Entwurf präsentiert, umgesetzt worden. Darüber hinaus sind zahlreiche Testfälle zur Qualitätssicherung vorhanden. Die Weitergabe an den Parser wurde über die Implementierung von Puffern gesichert. Durch diese Umsetzung ist ein nebenläufiger Scanvorgang möglich.

6.6. Fazit

Der Scanner wurde den Anforderungen entsprechend umgesetzt. Nach und während der Umsetzung waren die qualitätssichernden Komponenten stets aktiv, womit folgende Auswertungen erreicht werden konnten.

Die Testabdeckung, welche durch *eCobertura* überprüft wurde, liegt bei 100%. Somit wird der komplette Quelltext des Scanners durch die Testfälle abgedeckt.

PMD gibt null Warnungen, wodurch die Effizienz des Codes maximiert werden konnte.

Ein einheitlicher Programmierstil ist ebenfalls vorhanden, da *Checkstyle* die Einhaltung dieses Stils überprüft und mit null Checkstylewarnungen dokumentiert. Programmierfehler sind laut *FindBugs* nicht vorhanden, diese Aussage wird durch das fehlerfreie Durchlaufen aller Testfälle bestätigt.

7. Parser

7.1. Einführung

In diesem Kapitel wird die Arbeitsweise des Parsens erläutert. Dazu wurde ein Parser entwickelt, der die vom Scanner eingelesenen Konfigurationsdateien erhält und weiter verarbeitet. Aus diesen Dateien sollen nach dem Parse-Vorgang die zuvor modellierten Objekte erstellt werden. Diese werden letztendlich vom Printer weiterverarbeitet.

Zur Koordination zwischen Scanner und Parser ist es unerlässlich, sich auf eine Möglichkeit der Datenstromübertragung festzulegen.

7.2. Analyse

7.2.1. Anforderungen

Da die Symbole vom Scanner über Puffer übermittelt werden, muss der Parser auf die Puffer zugreifen können. Des Weiteren muss eine Schnittstelle definiert werden, über die auf die Pakete, die durch den Parser erstellt werden, zugegriffen werden kann. Die Pakete sollen gemäß der Definition der Konfigurationsgruppe erstellt und Fehlerfälle adäquat behandelt werden.

7.2.2. Ausgangslage

Ausgangslage sind die erarbeiteten Ergebnisse bezüglich der Konfigurationsdateien von fli4. Wichtigste Erkenntnis dabei war, dass die Konfiguration paketspezifisch erfolgt; dabei sind drei Dateien wichtig, in denen die Variablendefinitionen, die Typen sowie die Wertzuweisungen gespeichert sind. Diese sind in Tabelle 16 gelistet.

Art	Dateipfad
Variablendefinitionen	/check/<Paketname>.txt
Typen	/check/<Paketname>.exp
Wertzuweisungen	/config/<Paketname>.txt

Tabelle 16: Konfigurationsdateien von fli4l

Die Ergebnisse der Analyse im Verlauf dieses Projektteils wurden in ein Fachklassenmodell gegossen, das die Basis für die weiteren Projektphasen bereitstellt.

7.2.3. Fachklassendiagramm (ursprüngliche Version)

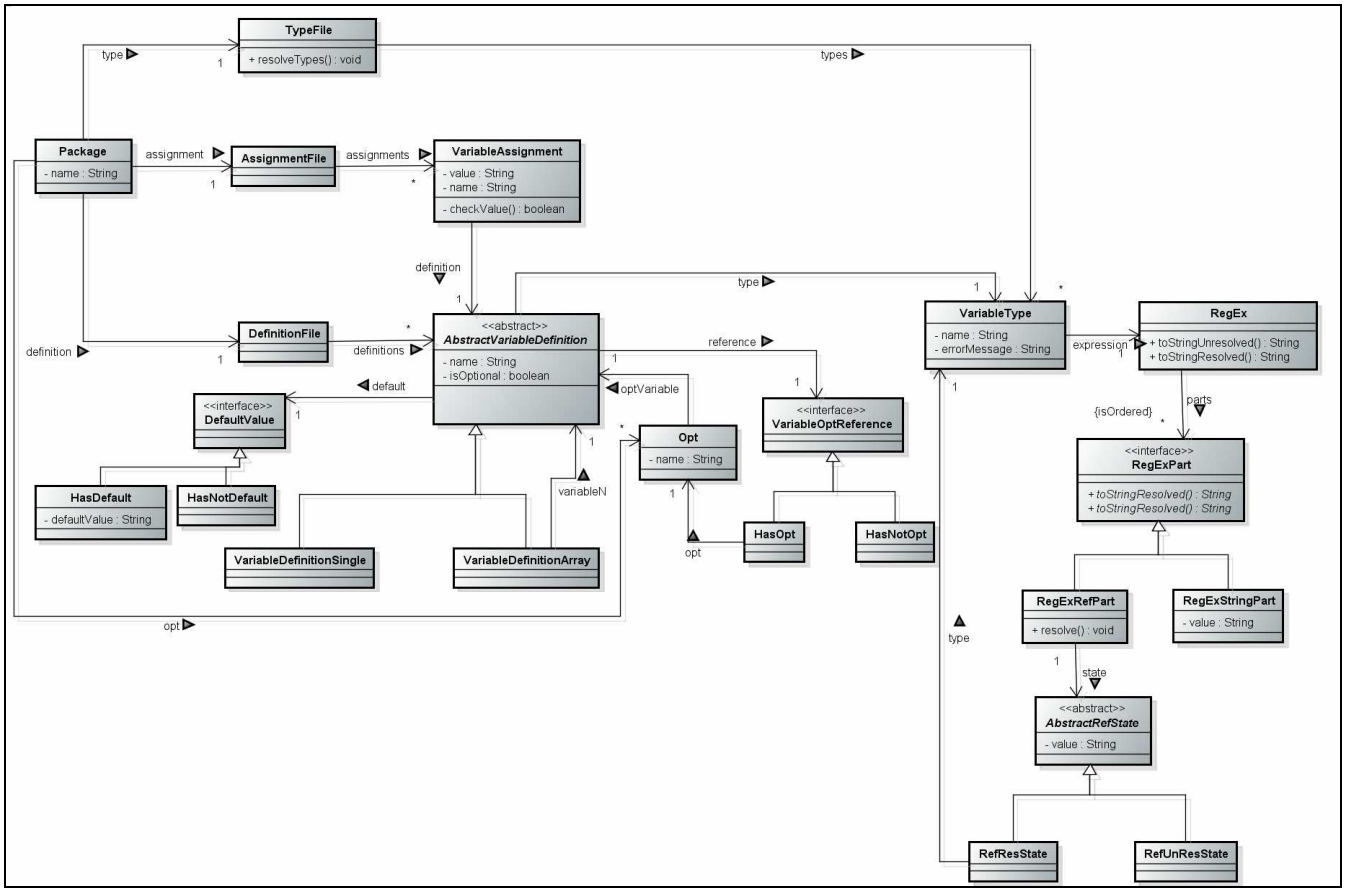


Abbildung 54: Alte Version des Klassendiagramms des Modells aus der Gruppe Konfiguration 1

7.2.4. Datenstromübertragung

Gemeinsam mit der Scanner-Gruppe musste sich auf eine Möglichkeit geeinigt werden, wie die eingelesenen Dateien an den Parser übergeben werden sollen. Für jede der drei Konfigurationsdateien werden dabei jeweils ein Scanner und der dazu passende Parser bereitgestellt. Die Schnittstelle zwischen den Scannern und Parsern sind wiederum drei Puffer.

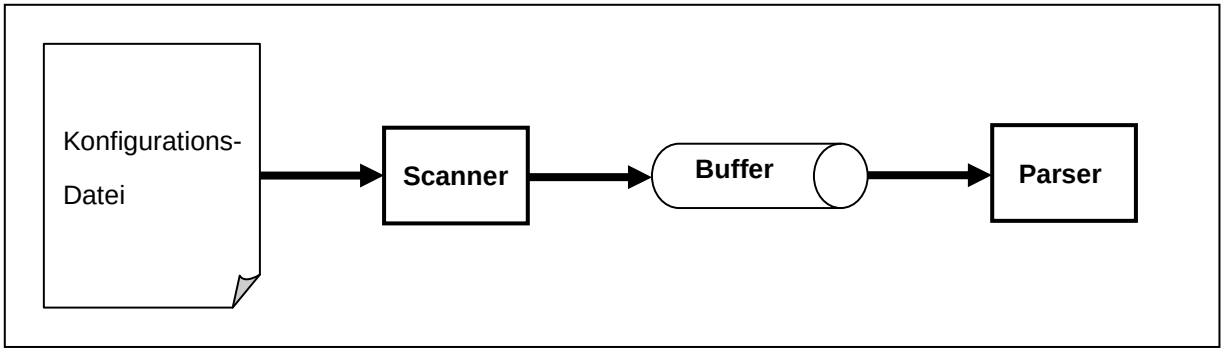


Abbildung 55: Datenfluss von der Konfigurationsdatei bis zum Parser

Die eingelesenen Zeichen innerhalb der Dateien werden vom Scanner im Puffer als Symbole abgelegt (z.B. RegExSymbol, StringSymbol). Mit diesem Puffer wird dann der jeweilige Parser aufgerufen.

7.2.5. Zustandsdiagramme der Status

Bei der Verarbeitung der Symbole im Parser wurde das Zustandsmuster (State-Pattern) benutzt. In Abhängigkeit der jeweiligen Symbole wird der Zustand des Parsers umgestellt. Die konkrete Behandlung der Symbole findet dann in dem entsprechenden Zustand statt. Da der Aufbau der einzelnen Konfigurationsdateien nicht identisch ist, wurde in jedem Parser das Zustandsmuster unterschiedlich implementiert, abhängig von der jeweiligen Konfigurationsdatei.

In jedem Parser existiert ein InitialState als Ausgangszustand, der zwischen Kommentaren und dem funktionalen Inhalt unterscheidet. Wurden alle Zustände des Kommentars bzw. funktionalen Inhalts durchlaufen, wird in den InitialState zurückgekehrt – damit ist der Kommentar bzw. die Definition fertig geparst.

Assignment States

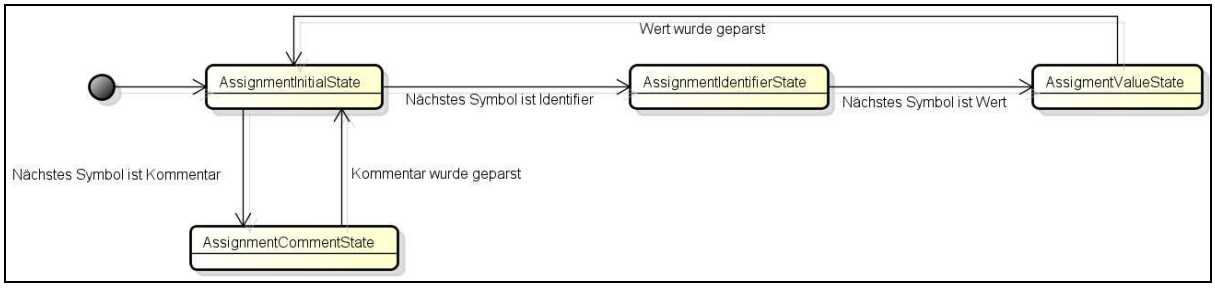


Abbildung 56: Zustandsdiagramm des AssignmentParsers

Da ein AssignmentFile nur aus Variablen und Zuweisungen besteht, werden auch nur zwei Zustände zum Parsen benötigt: der AssignmentIdentifierState für die Variable (z.B. ‚Hostname') und der AssignmentValueState für die zugehörige Zuweisung (z.B. ‚fli4l'). Das Gleichheitszeichen zwischen Variable und Zuweisung wird durch ein EqualSignSymbol repräsentiert.

Type States

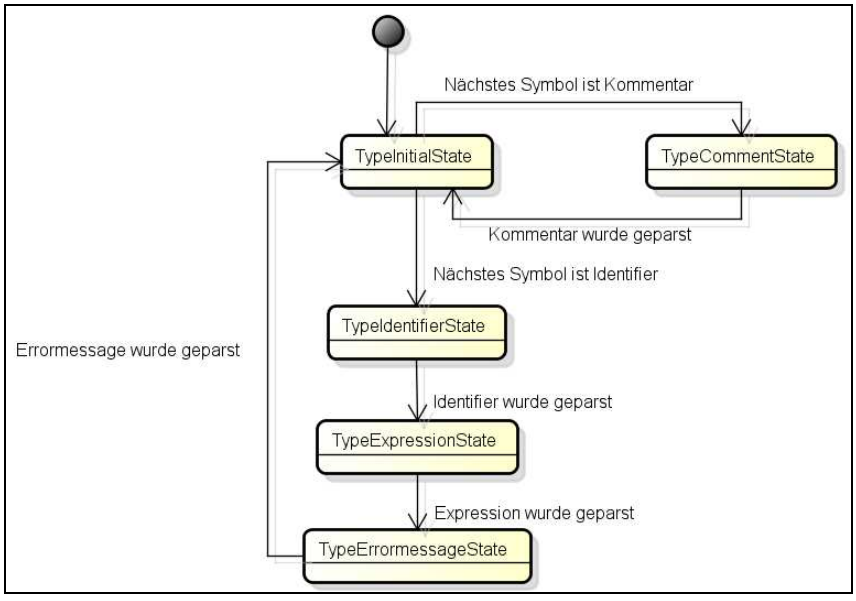


Abbildung 57: Zustandsdiagramm des TypeParsers

Eine Typendeklaration besteht aus Name, regulärem Ausdruck und Fehlermeldung. Ähnlich wie bei dem AssignmentFile wird nach dem Parsen des Namens im TypeIdentifierState auf die Existenz des EqualSignSymbols geprüft, das die Behandlung des darauf folgenden regulären Ausdrucks im TypeExpressionState einführt. Anschließend führt die Überprüfung eines vorhandenen Doppelpunktes (ColonSignSymbol) zur Verarbeitung der zugehörigen Fehlermeldung im ErrorMessageState.

Definition States

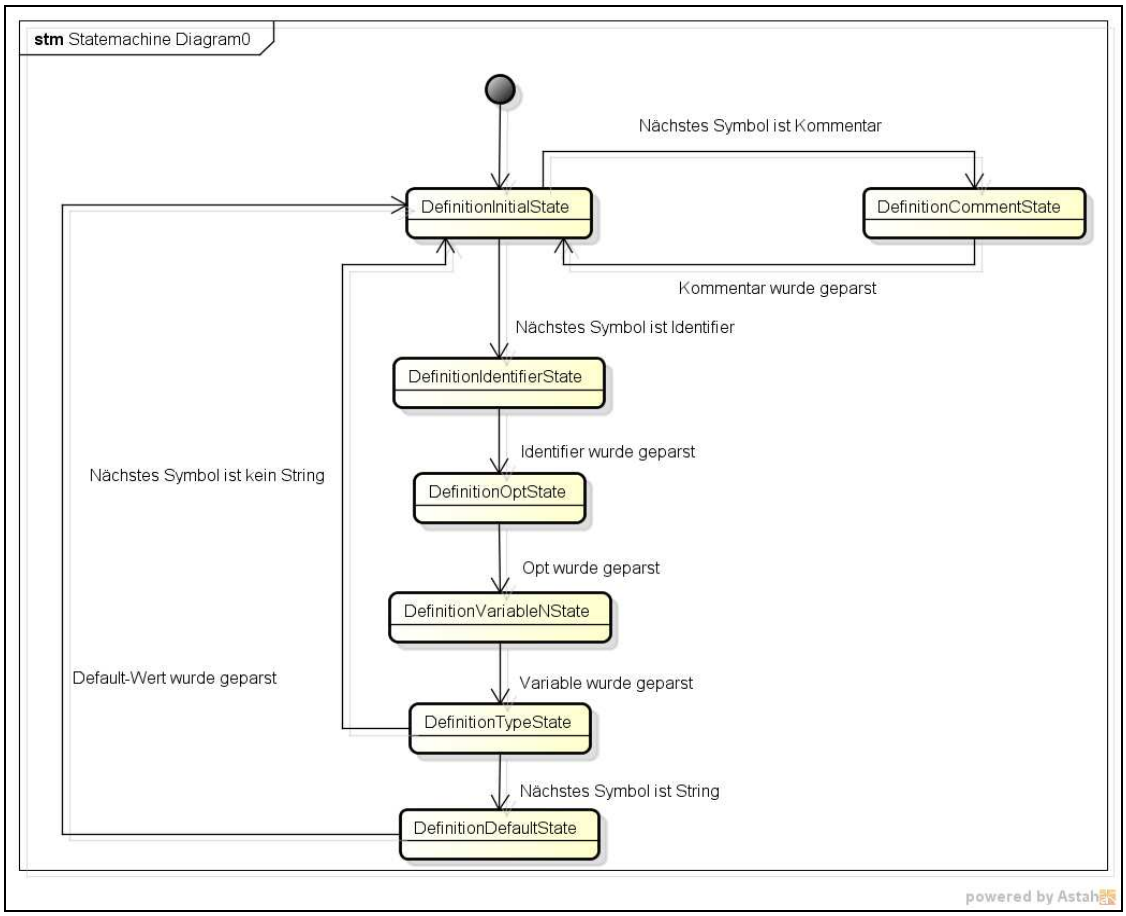


Abbildung 58: Zustandsdiagramm des DefinitionParsers

Die Zustände, die zum Parsen der Variablendefinition benötigt werden, orientieren sich an deren Aufbau: Nach dem Parsen der Variable im DefinitionIdentifierState wird die Opt_Variable im DefinitionOptState behandelt, indem auf die Existenz eines weiteren IdentifierSymbols geprüft wird. Existiert dieses, ist eine Opt_Variable vorhanden und wird geparst (Details siehe Kapitel 0 „Opts“). Die Variable_N wird danach im DefinitionVariableNState behandelt (siehe Kapitel 0 „VariableN“) und der Value im DefinitionTypeState. Eine Überprüfung untersucht, ob der Value ein regulärer Ausdruck oder eine Variable ist, und leitet den entsprechenden Parse-Vorgang ein. Der Standardwert wird schließlich im DefinitionDefaultState behandelt. Da der Standardwert nicht zwingend vorhanden sein muss, kann bereits vom DefinitionTypeState aus in den InitalState zurückgekehrt werden.

7.2.6. Klassendiagramm (ergänzte Version)

Im Zuge der Analyse musste das bisherige Klassenmodell angepasst und um verschiedene Klassen erweitert werden, damit die korrekte Behandlung der Konfigurationsdateien gewährleistet ist.

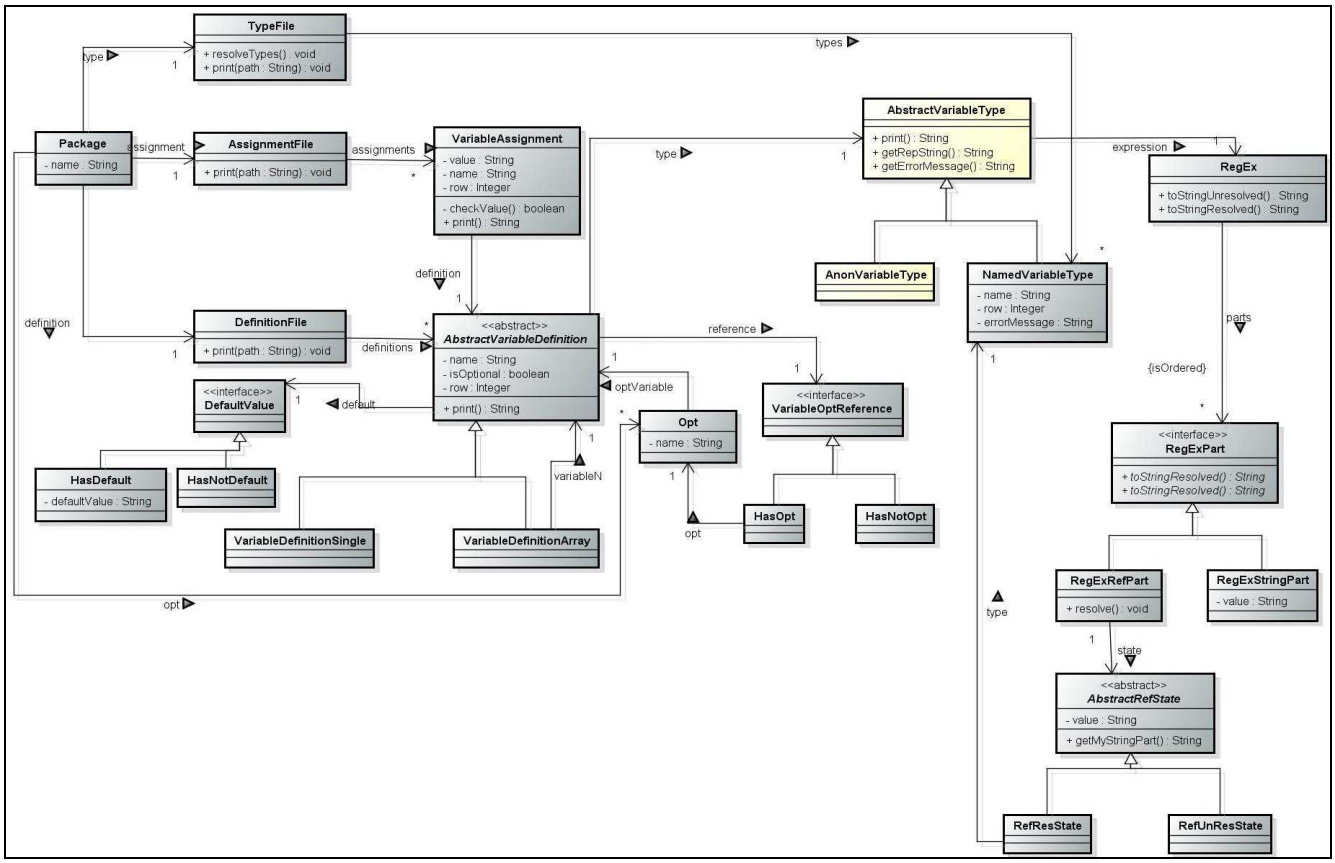


Abbildung 59: Ergänzte Version des Klassendiagramms des Konfigurationsmodells

Bei der Variablendefinition in der Spalte „Value“ ist die Wertzuweisung auf zwei Arten möglich: Entweder durch Nennung des konkreten Typs oder durch die direkte Angabe eines regulären Ausdrucks. Diese Unterscheidung muss beim Parsen berücksichtigt werden.

Aus diesem Grund wurde der ursprüngliche VariableType ersetzt durch den AnonVariableType (für die direkte Angabe des regulären Ausdrucks) und den NamedVariableType (der konkrete Typ).

7.3. Entwurf und Umsetzung

7.3.1. Klassendiagramm der Parser

Ergebnis der Analysephase ist ein Klassenmodell, das die komplette Struktur des Parsers beschreibt. Dabei werden die drei Parser, die für die einzelnen Konfigurationsdateien zuständig sind, mit ihren jeweiligen erreichbaren Zuständen in Verbindung gebracht.

In der Klasse GlobalFunctions befinden sich unterschiedliche Methoden, die von allen Zuständen benutzt werden und daher an eine zentrale Stelle ausgelagert wurden.

Der PackageParser dient zum Aufruf der einzelnen Parser und generiert am Ende aus den Symbolen, die er von ihnen erhält, ein fertig geparstes Package.

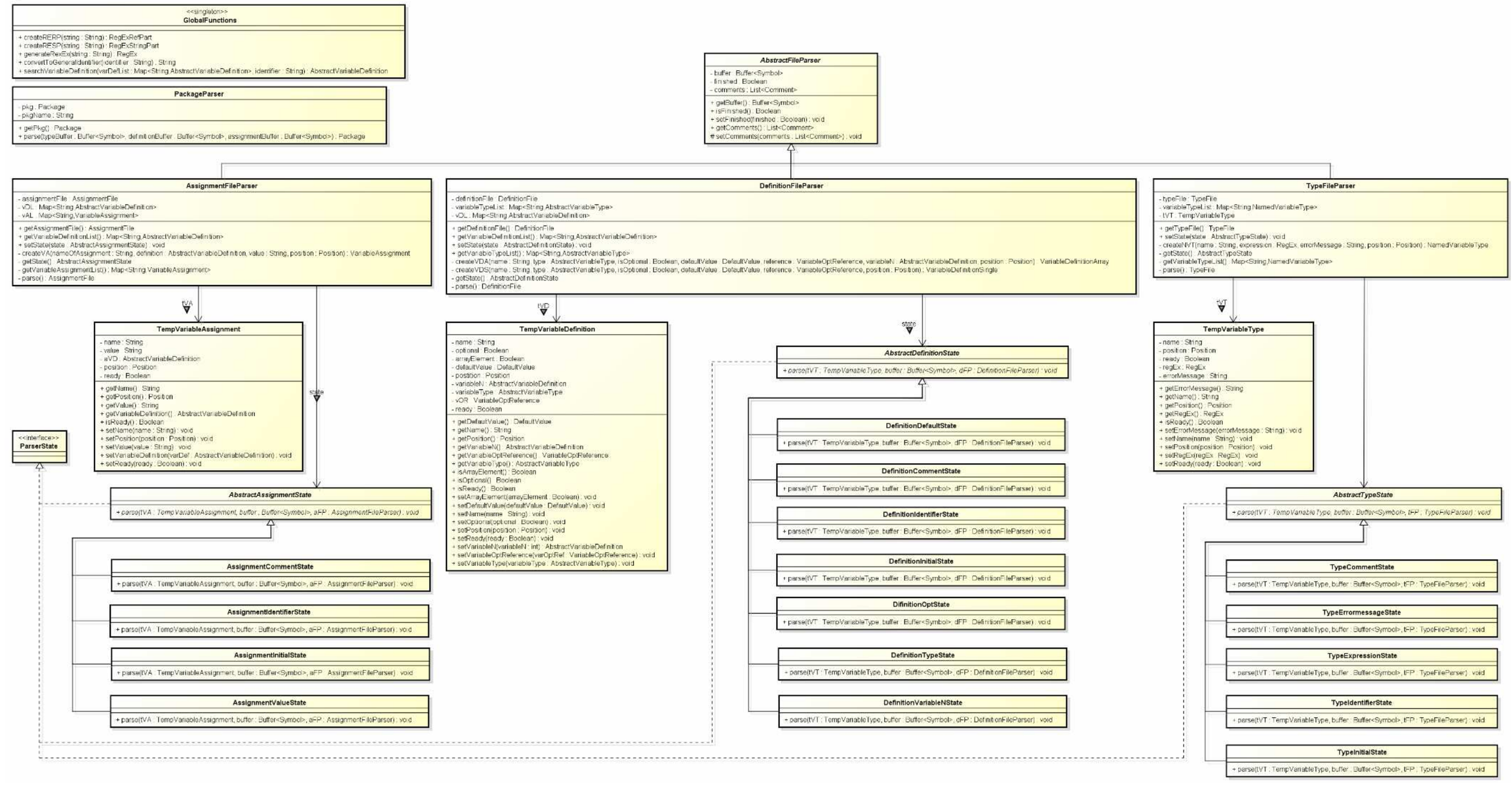

«singleton»
GlobalFunctions
+ createRERP(string : String) : RegExRefPart
+ createRESP(string : String) : RegExStringPart
+ generateRexEx(string : String) : RegEx
+ convertToGeneralIdentifier(identifier : String) : String
+ searchVariableDefinition(varDefList : Map<String,AbstractVariableDefinition>, identifier : String) : AbstractVariableDefinition
PackageParser
- pkg : Package
- pkgName : String
+ getPkg() : Package
+ parse(typeBuffer : Buffer<Symbol>, definitionBuffer : Buffer<Symbol>, assignmentBuffer : Buffer<Symbol>) : Package
AbstractFileParser
- buffer : Buffer<Symbol>
- finished : Boolean
- comments : List<Comment>
+ getBuffer() : Buffer<Symbol>
+ isFinished() : Boolean
+ setFinished(finished : Boolean) : void
+ getComments() : List<Comment>
# setComments(comments : List<Comment>) : void
AssignmentFileParser
- assignmentFile : AssignmentFile
- vDL : Map<String,AbstractVariableDefinition>
- vAL : Map<String,VariableAssignment>
+ getAssignmentFile() : AssignmentFile
+ getVariableDefinitionList() : Map<String,AbstractVariableDefinition>
+ setState(state : AbstractAssignmentState) : void
- createVA(nameOfAssignment : String, definition : AbstractVariableDefinition, value : String, position : Position) : VariableAssignment
- getState() : AbstractAssignmentState
- getVariableAssignmentList() : Map<String,VariableAssignment>
- parse() : AssignmentFile
DefinitionFileParser
- definitionFile : DefinitionFile
- variableTypeList : Map<String,AbstractVariableType>
- vDL : Map<String,AbstractVariableDefinition>
+ getDefinitionFile() : DefinitionFile
+ getVariableDefinitionList() : Map<String,AbstractVariableDefinition>
+ setState(state : AbstractDefinitionState) : void
+ getVariableTypeList() : Map<String,AbstractVariableType>
- createVDA(name : String, type : AbstractVariableType, isOptional : Boolean, defaultValue : DefaultValue, reference : VariableOptReference, variableN : AbstractVariableDefinition, position : Position) : VariableDefinitionArray
- createVDS(name : String, type : AbstractVariableType, isOptional : Boolean, defaultValue : DefaultValue, reference : VariableOptReference, position : Position) : VariableDefinitionSingle
- getState() : AbstractDefinitionState
- parse() : DefinitionFile
TypeFileParser
- typeFile : TypeFile
- variableTypeList : Map<String,NamedVariableType>
- tVT : TempVariableType
+ getTypeFile() : TypeFile
+ setState(state : AbstractTypeState) : void
- createNVT(name : String, expression : RegEx, errorMessage : String, position : Position) : NamedVariableType
- getState() : AbstractTypeState
- getVariableTypeList() : Map<String,NamedVariableType>
- parse() : TypeFile
tVA
tVD
tVT
state
TempVariableAssignment
- name : String
- value : String
- aVD : AbstractVariableDefinition
- position : Position
- ready : Boolean
+ getName() : String
+ getPosition() : Position
+ getValue() : String
+ getVariableDefinition() : AbstractVariableDefinition
+ isReady() : Boolean
+ setName(name : String) : void
+ setPosition(position : Position) : void
+ setValue(value : String) : void
+ setVariableDefinition(varDef : AbstractVariableDefinition) : void
+ setReady(ready : Boolean) : void
TempVariableDefinition
- name : String
- optional : Boolean
- arrayElement : Boolean
- defaultValue : DefaultValue
- position : Position
- variableN : AbstractVariableDefinition
- variableType : AbstractVariableType
- vOR : VariableOptReference
- ready : Boolean
+ getDefaultValue() : DefaultValue
+ getName() : String
+ getPosition() : Position
+ getVariableN() : AbstractVariableDefinition
+ getVariableOptReference() : VariableOptReference
+ getVariableType() : AbstractVariableType
+ isArrayElement() : Boolean
+ isOptional() : Boolean
+ isReady() : Boolean
+ setArrayElement(arrayElement : Boolean) : void
+ setDefaultValue(defaultValue : DefaultValue) : void
+ setName(name : String) : void
+ setOptional(optional : Boolean) : void
+ setPosition(position : Position) : void
+ setReady(ready : Boolean) : void
+ setVariableN(variableN : int) : AbstractVariableDefinition
+ setVariableOptReference(varOptRef : VariableOptReference) : void
+ setVariableType(variableType : AbstractVariableType) : void
TempVariableType
- name : String
- position : Position
- ready : Boolean
- regEx : RegEx
- errorMessage : String
+ getErrorMessage() : String
+ getName() : String
+ getPosition() : Position
+ getRegEx() : RegEx
+ isReady() : Boolean
+ setErrorMessage(errorMessage : String) : void
+ setName(name : String) : void
+ setPosition(position : Position) : void
+ setRegEx(regEx : RegEx) : void
+ setReady(ready : Boolean) : void
«interface»
ParserState
AbstractAssignmentState
+ parse(tVA : TempVariableAssignment, buffer : Buffer<Symbol>, aFP : AssignmentFileParser) : void
AssignmentCommentState
+ parse(tVA : TempVariableAssignment, buffer : Buffer<Symbol>, aFP : AssignmentFileParser) : void
AssignmentIdentifierState
+ parse(tVA : TempVariableAssignment, buffer : Buffer<Symbol>, aFP : AssignmentFileParser) : void
AssignmentInitialState
+ parse(tVA : TempVariableAssignment, buffer : Buffer<Symbol>, aFP : AssignmentFileParser) : void
AssignmentValueState
+ parse(tVA : TempVariableAssignment, buffer : Buffer<Symbol>, aFP : AssignmentFileParser) : void
AbstractDefinitionState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionDefaultState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionCommentState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionIdentifierState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionInitialState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DifinitionOptState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionTypeState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionVariableNState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
AbstractTypeState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeCommentState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeErrormessageState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeExpressionState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeIdentifierState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeInitialState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void

Abbildung 60: Klassendiagramm aller Parser-Klassen

7.3.2. Funktionsweise der Parser

Im PackageParser werden die einzelnen Parser (AssignmentFileParser, DefinitionFileParser und TypeFileParser) angestoßen und bekommen dabei ihre jeweiligen Puffer mitgeliefert.

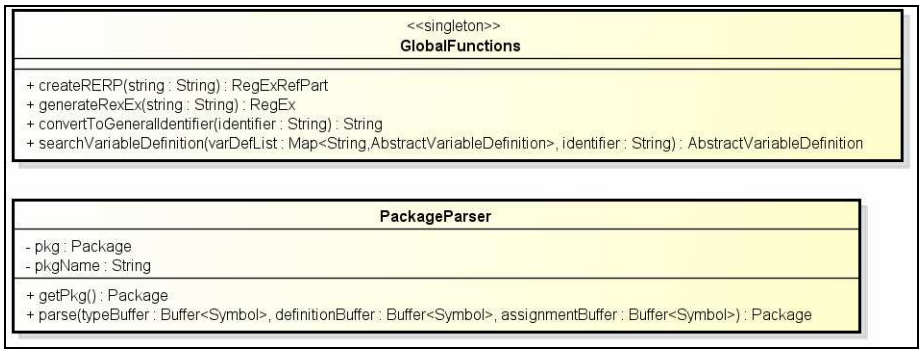


Abbildung 61: Klassen GlobalFunctions und PackageParser

Der Ablauf der einzelnen Parser ist im Grunde identisch und sei hier am Beispiel des TypeFileParsers erläutert: Im Konstruktor findet zunächst die Initialisierung der Variablen statt, anschließend wird der Parse-Vorgang des TypeFileParsers gestartet. Abhängig vom aktuellen Zustand des Parsers werden die Symbole, die dem Puffer entnommen werden, unterschiedlich behandelt – die jeweilige Behandlung wird in der Parse-Methode jedes einzelnen Zustandes durchgeführt.

Bei der Instanziierung eines Parsers wird der Zustand auf den jeweiligen InitialState gesetzt. Damit wird zuerst die Methode parse() des InitialStates aufgerufen. Gemäß der Zustandsdiagramme (siehe Kapitel 7.2.5) wird das Symbol aus dem Puffer auf das als nächstes erwartete Symbol (IdentifierSymbol oder CommentSymbol) überprüft. Liegt ein solches vor, so wird der Zustand des Parsers geändert und in den IdentifierState bzw. CommentState gewechselt.

Wechselt ein Parser in den CommentState, wird der Inhalt des Kommentars samt Position als Comment-Objekt in einer Liste abgespeichert. Danach wird jedes Mal wieder in den InitialState gewechselt.

Die Unterschiede in den Abläufen der jeweiligen Parser werden im Folgenden detailliert beschrieben.

TypeFileParser

Wurde in den TypeIdentifierState gewechselt, wird hier die Methode parse() ausgeführt. Von hier aus wird das nächste Symbol überprüft, ob es das erwartete Gleichheitszeichen (EqualSignSymbol) ist, das den darauf folgenden regulären Ausdruck einleitet (Behandlung siehe Kapitel 7.3.3).

Der Parse-Vorgang der Typ-Deklaration ist vollzogen, sobald die Parse-Methoden der als nächstes folgenden Zustände TypeExpressionState und TypeErrormessageState durchlaufen wurden. Im TypeErrormessageState wird der Zustand wieder auf TypeInitialState gesetzt, und das Parsen der nächsten Deklaration beginnt.

Sobald der Puffer geleert ist (das letzte Symbol ist ein EndSymbol), wird vom TypeFileParser ein neues TypeFile erzeugt. Dieses TypeFile wird wiederum an den PackageParser zurückgeliefert.

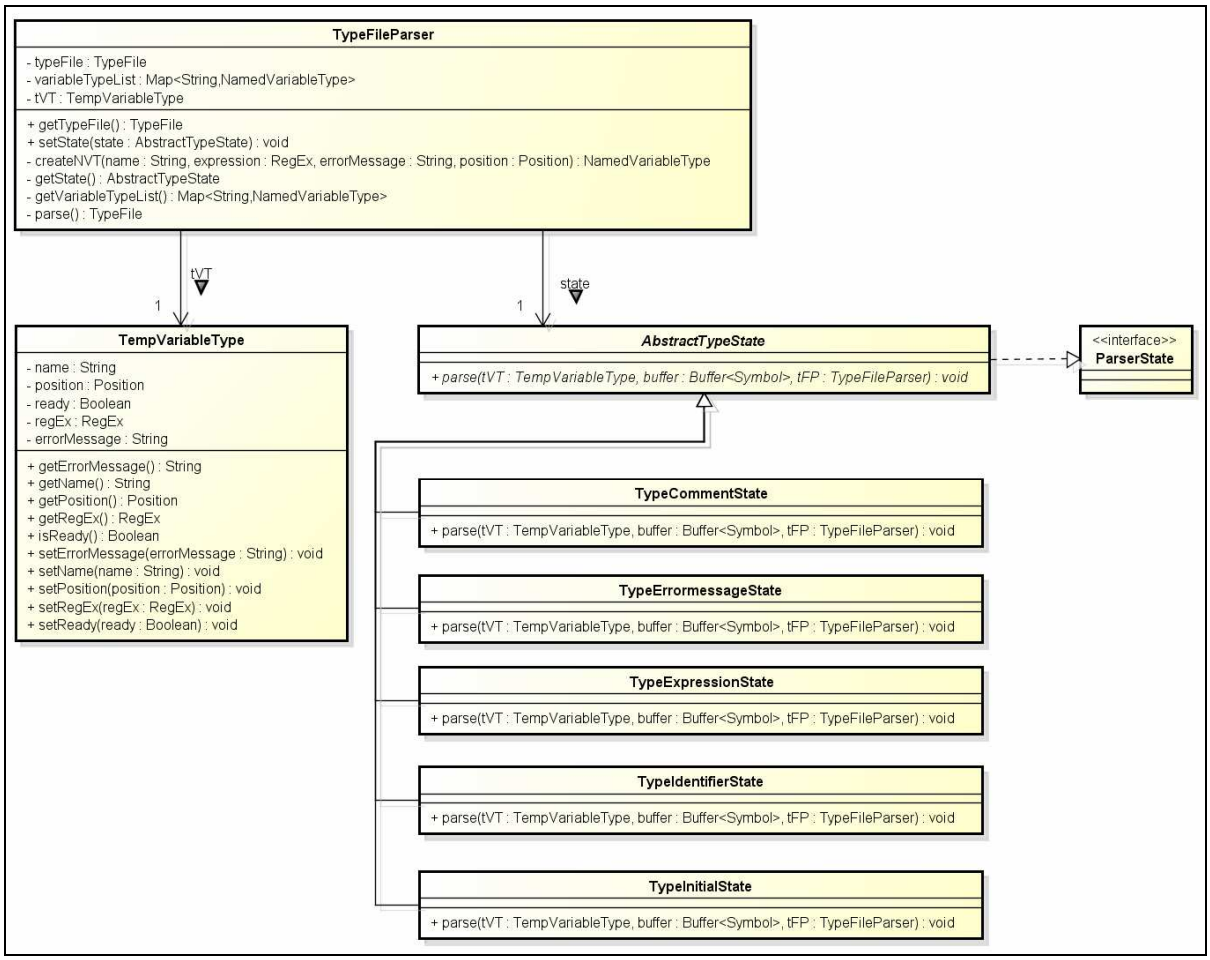

TypeFileParser
- typeFile : TypeFile
- variableTypeList : Map<String,NamedVariableType>
- tVT : TempVariableType
+ getTypeFile() : TypeFile
+ setState(state : AbstractTypeState) : void
- createNVT(name : String, expression : RegEx, errorMessage : String, position : Position) : NamedVariableType
- getState() : AbstractTypeState
- getVariableTypeList() : Map<String,NamedVariableType>
- parse() : TypeFile
tVT
1
state
1
TempVariableType
- name : String
- position : Position
- ready : Boolean
- regEx : RegEx
- errorMessage : String
+ getErrorMessage() : String
+ getName() : String
+ getPosition() : Position
+ getRegEx() : RegEx
+ isReady() : Boolean
+ setErrorMessage(errorMessage : String) : void
+ setName(name : String) : void
+ setPosition(position : Position) : void
+ setRegEx(regEx : RegEx) : void
+ setReady(ready : Boolean) : void
AbstractTypeState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
<<interface>>
ParserState
TypeCommentState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeErrormessageState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeExpressionState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeIdentifierState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void
TypeInitialState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, tFP : TypeFileParser) : void

Abbildung 62: Klassendiagramm des TypeFileParsers

DefinitionFileParser

Da ein Eintrag aus der Definitionsdatei bis zu fünf Einträge besitzt, werden auch im DefinitionsFileParser neben dem Initial- und dem CommentState bis zu fünf Zustände pro Eintrag durchlaufen.
Im DefinitionIdentifier State wird zunächst nur der Eintrag in der ersten Spalte ermittelt.

Im DefinitionOptState wird je nach ankommendem Symbol unterschieden, was geschehen soll. Enthält der Buffer ein EmptySymbol, so wird ein „HasNotOpt“ Objekt dem opt-Attribut hinzugefügt. Liegt jedoch ein IdentifierSymbol vor, wird aus der Definitionsliste die Definition mit entsprechendem Namen herausgesucht, in ein Opt-Objekt gebunden und dieses einem „HasOpt“ mitgegeben, welches dann dem opt-Attribut hinzugefügt wird.

Der DefinitionVariableNState unterscheidet ebenfalls zwischen EmptySymbol und IdentifierSymbol. Beim IdentifierSymbol ist zu beachten, dass die Definitionen der Array-Elemente im Namen anstatt der Ziffern %-Zeichen haben (IP_NET_1 und IP_NET_%). Wie die Umwandlung vorgenommen wird ist in Punkt 0 erläutert.

Im DefinitionTypeState kann entweder ein RegExpSymbol, welches als anonymer Typ der Definition angehängt wird, oder ein IdentifierSymbol, womit aus der Typen-Liste der entsprechende Typ ermittelt und angehängt wird, entgegengenommen werden. Die Zerlegung des RegEx verläuft dabei ebenso wie im TypeFileParser (erläutert in Kapitel 7.3.3).

Wird nach dem FileTypeState ein StringSymbol im Puffer übergeben, wird in den optionalen DefinitionDefaultState gewechselt. Hier wird ein Standardwert ermittelt.

Abschließend wird wieder in den DefinitionInitialState gewechselt.

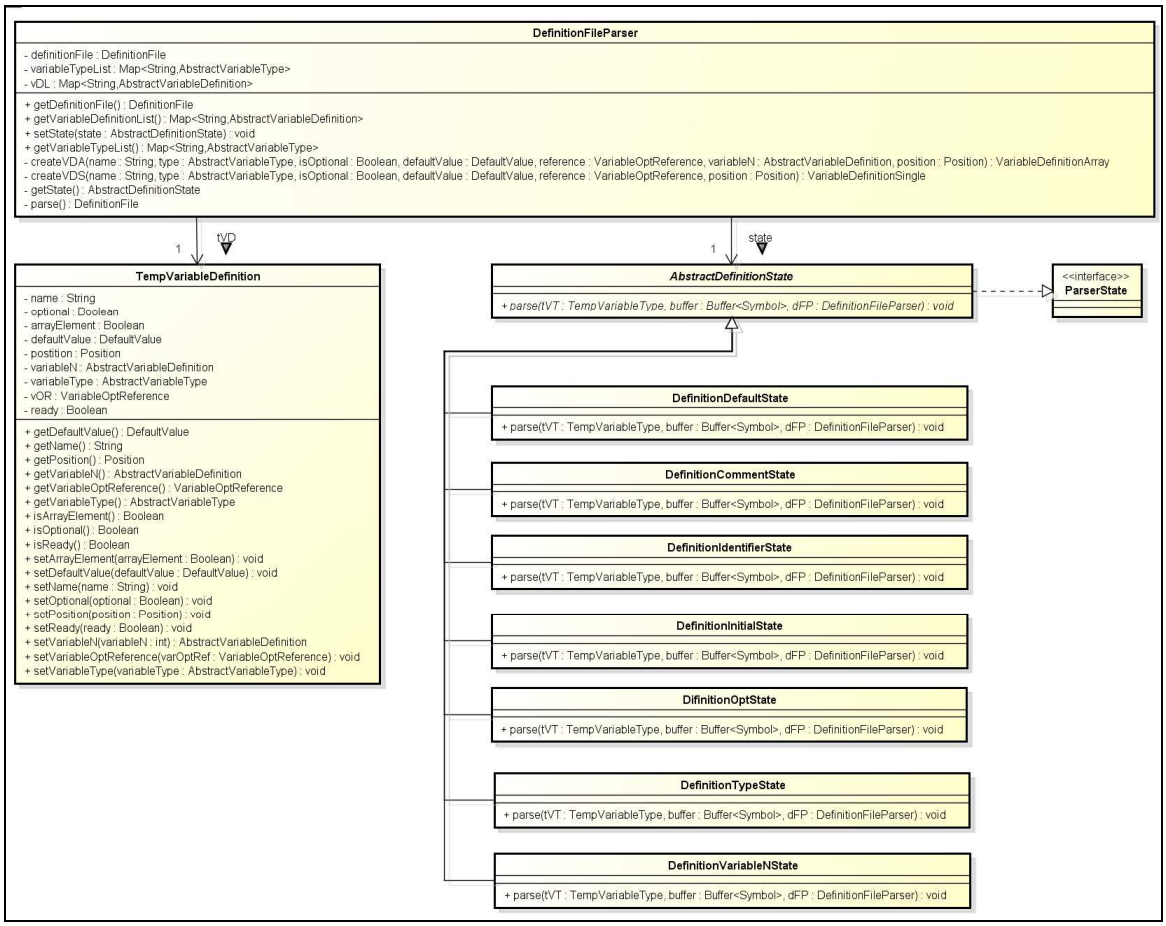

DefinitionFileParser
- definitionFile : DefinitionFile
- variableTypeList : Map<String,AbstractVariableType>
- vDL : Map<String,AbstractVariableDefinition>
+ getDefinitionFile() : DefinitionFile
+ getVariableDefinitionList() : Map<String,AbstractVariableDefinition>
+ setState(state : AbstractDefinitionState) : void
+ getVariableTypeList() : Map<String,AbstractVariableType>
- createVDA(name : String, type : AbstractVariableType, isOptional : Boolean, defaultValue : DefaultValue, reference : VariableOptReference, variableN : AbstractVariableDefinition, position : Position) : VariableDefinitionArray
- createVDS(name : String, type : AbstractVariableType, isOptional : Boolean, defaultValue : DefaultValue, reference : VariableOptReference, position : Position) : VariableDefinitionSingle
- getState() : AbstractDefinitionState
- parse() : DefinitionFile
1
tVD
1
state
TempVariableDefinition
- name : String
- optional : Boolean
- arrayElement : Boolean
- defaultValue : DefaultValue
- postition : Position
- variableN : AbstractVariableDefinition
- variableType : AbstractVariableType
- vOR : VariableOptReference
- ready : Boolean
+ getDefaultValue() : DefaultValue
+ getName() : String
+ getPosition() : Position
+ getVariableN() : AbstractVariableDefinition
+ getVariableOptReference() : VariableOptReference
+ getVariableType() : AbstractVariableType
+ isArrayElement() : Boolean
+ isOptional() : Boolean
+ isReady() : Boolean
+ setArrayElement(arrayElement : Boolean) : void
+ setDefaultValue(defaultValue : DefaultValue) : void
+ setName(name : String) : void
+ setOptional(optional : Boolean) : void
+ setPosition(position : Position) : void
+ setReady(ready : Boolean) : void
+ setVariableN(variableN : int) : AbstractVariableDefinition
+ setVariableOptReference(varOptRef : VariableOptReference) : void
+ setVariableType(variableType : AbstractVariableType) : void
AbstractDefinitionState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
<<interface>>
ParserState
DefinitionDefaultState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionCommentState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionIdentifierState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionInitialState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DifinitionOptState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionTypeState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void
DefinitionVariableNState
+ parse(tVT : TempVariableType, buffer : Buffer<Symbol>, dFP : DefinitionFileParser) : void

Abbildung 63:Klassendiagramm des DefinitionFileParsers

AssignmentFileParser

Im AssignmentIdentifierState wird auf einen Identifier überprüft und die zugehörige Definition in der VariableDefinitionList gesucht.

Das darauf folgende Gleichheitszeichen stößt den Wechsel in den AssignmentValueState an, in dem ein String erwartet wird. Damit ist der Parser einmal durch die Variablenzuweisung gelaufen und beginnt wieder von vorn im AssignmentInitialState.

Wurde über den Puffer das EndSymbol übermittelt, wird vom AssignmentFileParser ein neues AssignmentFile erzeugt und an den PackageParser zurückgeliefert.

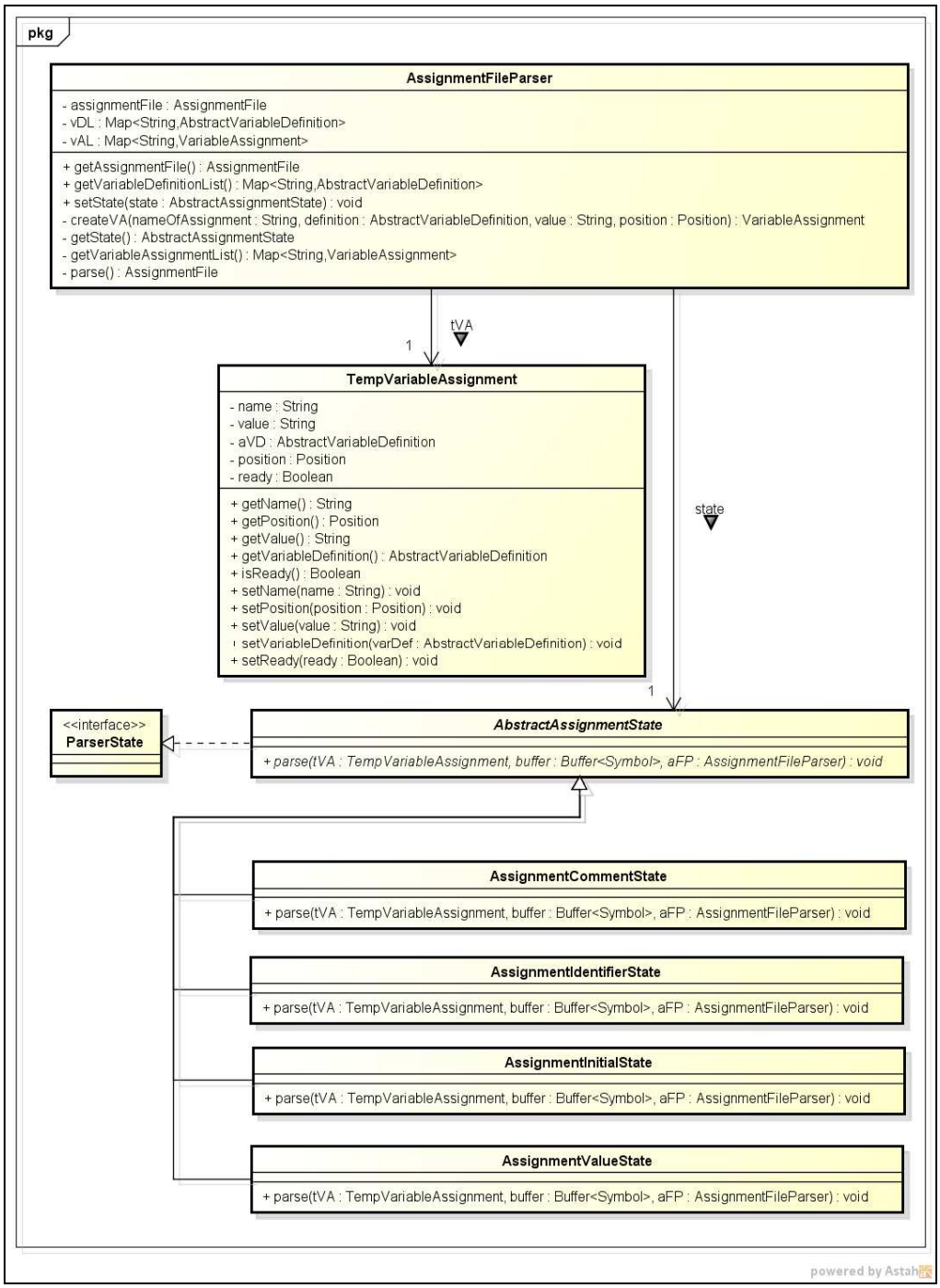


Abbildung 64: Klassendiagramm des AssignmentFileParsers

Im letzten Schritt erstellt der PackageParser aus dem TypeFile des TypeFileParsers, dem DefinitionFile des DefinitionFileParsers und dem AssignmentFile des AssignmentFileParsers ein neues Package.

7.3.3. Sonderbehandlungen

tempParserObjects

Um ein Objekt aus dem Modell aufzubauen, sind alle benötigten Attribute als Parameter im Konstruktor zu übergeben. Da diese Informationen jedoch erst im Laufe des Scannens der Symbole gesammelt werden, wird zu jedem dieser Objekte ein temporäres Objekt erzeugt, welches von State zu State weitergereicht wird, wo es dann stückweise mit allen benötigten Informationen befüllt werden

kann. Sind alle benötigten Attribute zusammengekommen, werden diese aus dem temporären Objekt ausgelesen und dem Konstruktor des Zielobjekts übergeben.

VariableN

Elemente eines Arrays werden mit „_%“ bzw „_%_“ im Variablennamen definiert. Die %-Zeichen werden dabei in der Zuweisungsdatei durch Zahlen ersetzt, die die Position des Elements im Array kennzeichnen. Um die Definition dazu zu finden, wurde die Methode searchVariableDefinition() und der darin verwendete convertToGeneralIdentifier() in der GlobalFunctions-Klasse implementiert. Diese ersetzen via RegEx mit den Klassen Pattern und Matcher aus dem Paket java.util.regex die Zahlen durch %-Zeichen und suchen dann in der Definitionsliste nach der Definition, um sie der Zuweisung als zugehörige Definition hinzuzufügen.

Opts

Um die Opts zu ermitteln, wird beim Parsen der Identifier in einer Definitionsdatei der Text auf den Beginn mit „OPT_“ überprüft. Trifft dies auf einen Identifier zu, wird ein neues Opt-Objekt erstellt und zu der Liste der Opts, die im Parser gespeichert wird, hinzugefügt. Wird nun in der Opt-Spalte der Definitionsdatei ein Identifier übergeben, wird in dieser Liste nach dem passenden Opt gesucht und dieses dem Konstruktor des Definitionsobjekts übergeben.

RegEx

Reguläre Ausdrücke werden vom Scanner als String übergeben. Dieser muss für das RegEx-Objekt aufgespalten werden. Ein RegEx in fli4l besteht aus den gewöhnlichen Elementen eines regulären Ausdrucks und aus Referenzen, die auf den Identifier eines anderen regulären Ausdrucks zeigen können. Für diese Aufspaltung wird in der GlobalFunctions-Klasse die Methode generateRegEx() verwendet. Diese spaltet den String in mehreren Schritten. Zunächst wird nach dem ersten Vorkommen des Patterns „(RE:“ gesucht. Dieses indiziert einen darauf folgenden Identifier und damit eine Referenz auf einen anderen regulären Ausdruck. Als nächstes wird nach dem darauf folgenden „)“ gesucht, da dort der Identifier zu Ende ist. So ist der String, sofern mindestens eine Referenz vorhanden war, in drei Teile aufgespaltet. Diese Methode wird so oft angewandt, bis der gesamte String in Referenzen und Nicht-Referenzen aufgespaltet ist. Die einzelnen Elemente werden in einer Liste gespeichert und dem VariableType bei der Instanziierung übergeben.

AnonymeVariableTypes

In neueren Versionen von fli4l ist es möglich, in die Definitionsdatei in der Typ-Spalte nicht nur den Identifier eines VariableTypes, sondern auch direkt einen regulären Ausdruck anzugeben. Dieses wird durch ein „RE:“ gekennzeichnet. Die darauf folgenden Zeichen werden dann wieder als String dem Parser übergeben, der daraus ein RegEx-Objekt parst. Da dieser Variablentyp jedoch weder Namen noch Fehlermeldung besitzt, wird ein anonymer VariableType (AnonVariableType) erzeugt, der direkt der Definition zugeordnet wird, jedoch nicht in der Liste der Variablentypen zu finden ist.

7.3.4. Fehlerbehandlung

Fehlerfälle werden mit der ParserException abgefangen: Dazu gehört das Auftreten eines unerwarteten Symbols und wenn eine Abhängigkeit zwischen den Klassen nicht erfüllt ist.

7.3.5. Testfälle

Zum Testen der Funktionalitäten wurden unterschiedliche Testfälle erstellt, darunter ein kompletter Aufbau eines AssignmentFiles mit anschließendem testmäßigen Durchlaufen der einzelnen Parser, sowie Testfälle zur Generierung von regulären Ausdrücken aus Zeichenketten und zur Konvertierung des Namens eines Array-Elements in seinen generalisierten Namen.

8. Printer

8.1. Einführung

Im Folgenden wird der Aufbau und Ablauf des Printers erläutert, der die von Scanner und Parser erzeugten Objekte ausgibt.

8.2. Analyse

Die Aufgabe des Printers ist es, die, durch Scanner und Parser erzeugte Modellwelt, entweder in einer txt-Datei oder einer exp-Datei auszugeben. Bei einer Definitionsdatei und einer Zuweisungsdatei wird ein txt-File ausgegeben. Bei der Typendatei wird ein exp-File ausgegeben. Der Name der jeweiligen Datei entspricht dem zugeordneten Paketnamen. Die Definition- und Typdateien werden unter dem Pfad fli4l\check\ abgelegt, die Zuweisungsdateien unter dem Pfad fli4l\config\.
Hierfür muss die übergebene Modellwelt genommen werden, eine Datei mit entsprechendem Namen und Endung erstellt und geöffnet werden und die Modellwelt zeilenweise in die entsprechende Datei ausgegeben werden. Zeilenweise insofern, dass die Einträge in der gleichen Reihenfolge wie beim Einlesen geschrieben werden. Neue Einträge sollen an das Ende geschrieben werden. Nach der erfolgreichen Ausgabe muss die erstellte Datei wieder geschlossen werden. Zu der Modellwelt gehören neben Einträgen (Typen, Definitionen und Zuweisungen) auch Kommentare, die in jeder Datei auftreten können.

Ein weiterer Punkt, der bei der Entwicklung eines Printers berücksichtigt werden muss, ist die Anordnung der einzelnen Elemente. Zum einen muss die Zeilenreihenfolge, die beim Parsen erstellt wird, berücksichtigt und die Zeilen in der entsprechenden Reihenfolge ausgegeben werden. Zum anderen muss man sich über die Anordnung innerhalb einer Zeile Gedanken machen. Würde man dies nicht tun, würde der zu druckende Inhalt ohne Leerzeichen, Hochkomma und Gleichheitszeichen aneinander gereiht werden. Die Ausgaben sollen aber den bestehenden fli4l-Dateien entsprechen, daher muss zunächst geklärt werden wie Definitionsdatei, Zuweisungsdatei und Typdatei aufgebaut und somit gedruckt werden müssen.

8.2.1. Definitionsdatei (Definitionfile)

Im Definitionfile werden die Variablen definiert. Das Definitionfile besteht aus vier Spalten, die der Printer abbilden muss. Die Besonderheit liegt auf der zweiten und dritten Spalte, da diese nicht immer mit einer Referenz gefüllt sind. In dem Fall, dass keine Referenz enthalten ist, muss der Platzhalter „-“ abgebildet werden.

```
CONSOLE_BLANK_TIME      -                       -                       ENUMERIC
BEEP                    -                       -                       YESNO
SER_CONSOLE             -                       -                       CONSOLE
SER_CONSOLE_IF          SER_CONSOLE             -                       NUMERIC
SER_CONSOLE_RATE        SER_CONSOLE             -                       NUMERIC
NET_DRV_N               -                       -                       NUMERIC
NET_DRV_%               -                       NET_DRV_N               NET_DRV
NET_DRV_%_OPTION        -                       NET_DRV_N               NONE
```

Abbildung 65: Variablen Definition in fli4l

8.2.2. Zuweisungsdatei (Assignmentfile)

Im Assignmentfile werden die Variablen mit Werten gefüllt. Zunächst werden die Variablen zugewiesen. Hierfür wird der Variablenname und nach einem Gleichheitszeichen der Variableninhaltangegeben. Nach der Variablenzuweisung können mit dem #-Symbol Kommentare eingeleitet werden. Im Assignmentfile ist es auch möglich in einer Zeile nur einen Kommentar zu schreiben. Im base-Package ist diese Anordnung von Variablenzuweisung und Kommentar auf zwei Spalten verteilt. In der ersten Spalte ist die eigentliche Zuweisung zu finden und in der zweiten Spalte ist der Kommentar angeordnet.

```
HOSTNAME='fli4l'            # name of fli4l router
PASSWORD='fli4l'            # password for root login (console, sshd, imond)
BOOT_TYPE='cd'              # boot device: fd, fdx2, dualfd, hd, cd, ls120,
                            # integrated, attached, netboot, pxeboot
                            # (cd, cdemul, hd and ls120 need an
                            # installed opt hd)
MOUNT_BOOT='ro'             # mount boot device (floppy): ro, rw, no
BOOTMENU_TIME='5'           # waiting time of bootmenu in seconds
                            # to activate normal boot
```

Abbildung 66: Variablen Zuweisung in fli4l

8.2.3. Typendatei (Typefile)

Im Typefile werden die Wertebereiche durch reguläre Ausdrücke definiert. Dabei sieht die allgemeine Anordnung so aus, dass zunächst der Typname angegeben wird, dann, nach einem Gleichheitszeichen und mit Hochkommata gekennzeichnet der reguläre Ausdruck und zum Schluss, ebenfalls mit Hochkommata versehen und nach einem Doppelpunkt, die zugehörige Fehlermeldung (ErrorMessage).

Im Base-Package beispielsweise wird der Inhalt des Typefiles in drei Spalten angeordnet. Die Breite der Spalten ist fest vorgegeben und verschiebt sich nur, wenn der Spalteninhalt mehr Platz braucht. Eine Besonderheit bildet die dritte Spalte. Ist der Inhalt der zweiten Spalte zu lang, wird die Fehlermeldung (ErrorMessage) in den nächsten Absatz geschrieben und beginnt dort auf Höhe der der zweiten Spalte.

```
NONE            = '.*'                  : ''
NOTEMPTY        = '.*[^ ]+.*'           : 'should not be empty'
NOBLANK         = '[^ ]+'               : 'should not contain spaces'
ENOBLANK        = '()|(RE:NOBLANK)'     : 'should be empty or should not contain spaces'
NUMERIC         = '0|[1-9][0-9]*'       : 'should be numeric (decimal) without leading zero(s)'
ENUMERIC        = '()|(RE:NUMERIC)'     : 'should be numeric (decimal) without leading zero(s) or empty'
NUMERICLZ       = '[0-9][0-9]*'         : 'should be numeric (decimal) with or without leading zero(s)'
ENUMERICLZ      = '()|(RE:NUMERICLZ)'   : 'should be numeric (decimal) with or without leading zero(s) or empty'
DOT_NUMERIC     = '(RE:NUMERIC).[0-9]+'
                : 'should be numeric (decimal) with dot e.g. 5.0'
```

Abbildung 67: Variablen Typen in fli4l

8.2.4. Beachtenswertes

Kommentare können in jedem File auftreten, sie werden mit dem #-Symbol eingeleitet. Kommentare können nach einem Eintrag am Zeilenende, oder alleine in einer Zeile stehen. Beispielsweise kann eine Kopfzeile durch Kommentare realisiert werden.

8.3. Entwurf

Für die Umsetzung des Printers wird die in der Konfigurations-Gruppe entwickelte Struktur nur um einzelne Operationen erweitert:

In den Klassen „AssignmentFile“, „DefinitionFile“, „TypeFile“, „VariableType“, „VariableAssignment“, „VariableDefinitionSingle“, „VariableDefinitionArray“ und “Comment” wird jeweils die Operation print() implementiert. “Comment“ ist eine Klasse zur Repräsentation von Kommentaren in der Modellwelt. „Comment“ sowie die drei Eintragsklassen haben ein neues Attribut „Position“ bekommen, das die Position des Eintrags bzw. Kommentars repräsentiert. In den Klassen „AssignmentFile“, „DefinitionFile“ und „TypeFile“ wird durch die Einträge und Kommentare iteriert und es werden diese gedruckt. Bevor der nächste Eintrag eingelesen und ausgegeben wird, wird ein manueller Zeilenumbruch eingefügt, damit auch die Ausgabe eine zeilenweise Anordnung erhält. In den Klassen „VariableDefinitionSingle“, „VariableDefinitionArray „VariableType“ und „VariableAssignment“ definiert die print()-Methode die Anordnung der einzelnen Elemente innerhalb der Zeilen und gibt dieses an die zugehörige print()-Methode in den File-Klassen. In den Klassendiagrammen haben sich folgende Änderungen ergeben.

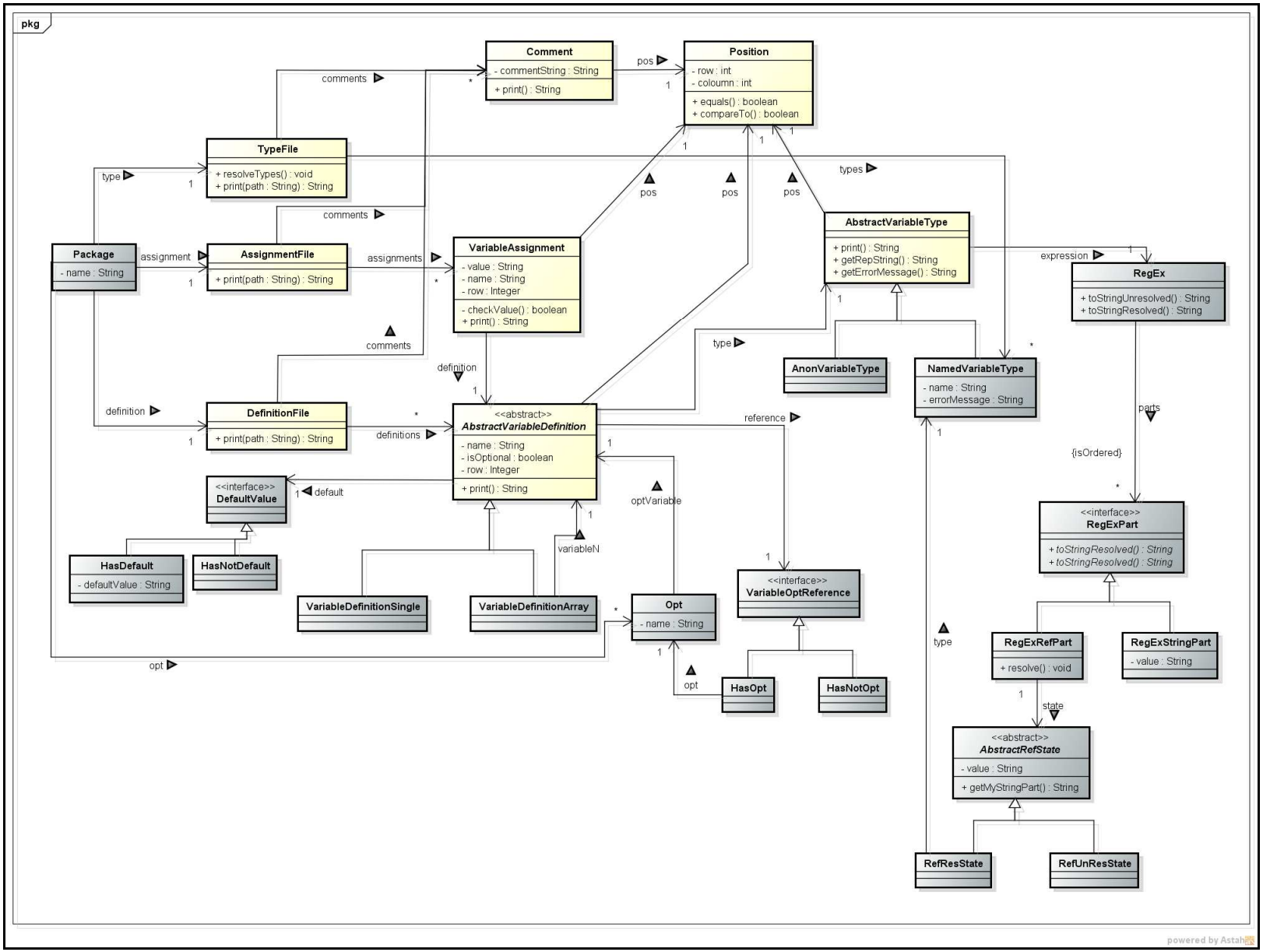


Abbildung 68: Änderungen im Klassendiagramm

8.4. Implementierung

Die Implementierung selbst ist durch den Quelltext und die Javadoc-Kommentare beschrieben. An dieser Stelle soll jedoch der Druck-Algorithmus mit Hilfe natürlicher Sprache beschrieben und auf einzelne Angelegenheiten wie das Drucken von Dateien an sich eingegangen werden.

8.4.1. Der Druckalgorithmus

1. Vorbereiten des zu druckenden Strings

   - Die Positionen von Einträgen und Kommentaren in einer Liste sortieren.
   - Durch die Liste iterieren und auf den jeweiligen Einträgen bzw. Kommentaren die print()-Methode aufrufen. Dadurch wird ein StringBuffer aufgebaut.

2. Das Drucken des StringBuffers in eine Datei.

8.4.2. Das Drucken des StringBuffers

Bei dem im Entwurf beschriebenen Iterieren wird zuerst in einen java.util.StringBuffer geschrieben. Nach dem Zusammensetzen der zu druckenden Zeilen in den StringBuffer wird dieser mit einem einzigen writer.write(buffer.toString()) geschrieben. So ist das tatsächliche Drucken an einer Stelle zentriert.

Zum Drucken einer Datei wurde auf folgende Klassen der Java-Bibliothek zurückgegriffen:

- ➔ OutputStreamWriter: Ein “Schreiber” für Dateien, initialisiert mit einem FileOutputStream und einem encoding.
- ➔ FileOutputStream: Das „File“ in das geschrieben wird, initialisiert mit dem Dateipfad(wird angelegt wenn nicht vorhanden).

FileOutputStream

```
try {
	final OutputStreamWriter writer =
			new OutputStreamWriter(new FileOutputStream(pathToPrint),
					basic.PrinterConstants.ENCODING);
	try {
		writer.write(stringBuffer.toString());
	} finally {
		writer.close();
	}
} catch (final IOException ioEx) {
	thrownew PrinterException(basic.PrinterConstants.PRINTEXCEPTION, ioEx);
}
```

- ➔ Die write()-Operation schreibt den übergebenen String in das File.
- ➔ Die close()-Operation schließt den OutputStreamWriter und das erzeugte File.

8.4.3. Beachtenswertes

Eine mögliche Exception die beim Printer auftreten kann ist eine IOException. Die IOException kann beim Zugreifen über Streams, Dateien und Verzeichnisse auf Informationen auftreten. Die IOException wird abgefangen und als PrinterException weitergeworfen, die ursprüngliche Exception kann aber jederzeit aus der PrinterException extrahiert werden.

9. Pakete

9.1. Einführung

In diesem Kapitel wird die Analyse der Paketbeschreibungsdateien und der Darstellung ihrer Struktur durch ein objektorientiertes Modell behandelt. Außerdem wird die Erstellung von Parser und Scanner für Paketbeschreibungsdateien erläutert.

9.2. Grundlagen

Die Paketdatei, welche sich im Dateiordner „opt“ befindet enthält Anweisungen, welche die folgenden Fragen beantworten:

- Welche Dateien gehören zu welchem OPT[17]?
- Wann werden diese Dateien in das zu generierende „opt.img“ beziehungsweise „rootfs“ übernommen?
- Welche Rechte, „uid“ und „gid“ erhalten diese Dateien?
- Welche Konvertierungen vor Aufnahme in die „tar-Datei“ sollen erfolgen?

Darauf basierend generiert „mkfli4l“ eine Liste mit den Informationen, welche für das Erstellen des Tarballs[18] notwendig sind. Kommentare sowie leere Zeilen werden hierbei ignoriert. In der ersten Zeile dieser List sollte die Version des Opt-Dateiformates definiert werden. Diese Zeile muss wie folgt aufgebaut sein:

<Spalte 1>	<Spalte 2>	<Spalte 3>
Opt_format_version	1	-

Tabelle 17: Zeilenaufbau der ersten Spalte

Alle weiteren Zeilen liegen in folgender Syntax vor:

<Spalte 1>	<Spalte 2>	<Spalte 3>	<Folgende Spalten>
Variable	Wert	Datei	Optionen

Tabelle 18: Zeilenaufbau der folgenden Spalten

Die **erste Spalte** beinhaltet die Bezeichnung der Variablen. Diese Bezeichnung kann beliebig oft in der ersten Spalte auftauchen, falls mehreren Dateien von ihr abhängig sind.

Die **zweite Spalte** beinhaltet den Wert der Variable. Falls die in der ersten Spalte genannte Variable mit diesem Wert übereinstimmt und aktiv[19] ist, wird die Datei der dritten Spalte übernommen.

[17] Neben den beiden Komponenten Basispaket und Zusatzpaket gibt es so genannte optionale Module (OPTs). Einige dieser OPTs befinden sich bereits im Basispaket, andere wiederum können zusätzlich installiert werden.
[18] Tarballs enthalten neben schon kompilierten Binärdateien auch den zugehören Quellcode, welcher zusätzlich noch mit Konfigurationsdateien für die Installation und der zugehörigen Dokumentation angereichert wird.
[19] Das bedeutet, falls eine Variable von einer opt-Variablen abhängt, muss diese den Wert „yes“ haben.

In der **dritten Spalte** steht der Name der zu übernehmenden Datei, deren Dateipfad relativ zum opt-Verzeichnis angegeben ist. Voraussetzung ist hierbei die Auslesbarkeit und Existenz der Datei. Die folgenden vier Besonderheiten dieses Spaltenwertes sind zu beachten:

1. Ist ein „rootfs:“- beziehungsweise „rootfs:files“- Präfix vorhanden, wird dieses entfernt und die Datei in die Liste der in das rootfs aufzunehmenden Dateien übernommen.

2. Befindet sich die Datei unterhalb eines aktuellen Konfigurationsverzeichnisses, wird diese in die Liste der aus diesem Verzeichnis zu übernehmenden Dateien aufgenommen.

3. Liegt die Datei nicht unterhalb eines aktuellen Konfigurationsverzeichnisses, wird diejenige Datei verwendet, welche unter opt existiert.

4. Ist die Datei ein Kernel-Modul, kann die konkrete Kernel-Version durch „$KERNEL_VERSION“ ersetzt werden. Vorteil dieser Methode ist, dass einem Paket mehrere Module für verschiedene Kernversionen mitgegeben werden können, wobei immer die für den Kern richtige Version mit kopiert wird.

In den **folgenden Spalten** können Optionen für die Konvertierung, Rechte der Dateien, Eigentümer und Gruppen stehen. Die folgende Abbildung zeigt die möglichen Optionen.

Option	Bedeutung	Standardwert
type	Der Typ des Eintrags: *local* Dateisystem-Objekt *file* Datei *dir* Verzeichnis *node* Gerät *symlink* (symbolische) Verknüpfung Wenn vorhanden, muss diese Option an erster Stelle stehen. Der Typ “local” steht hierbei für den Typ eines im Dateisystem existierenden Objekts und entspricht somit “file”, “dir”, “node” oder “symlink” (je nachdem). Die anderen Typen mit Ausnahme von “file” können Einträge im Archiv erzeugen, die nicht im lokalen Dateisystem vorliegen müssen. Das wird z.B. benutzt, um Gerätedateien im ROOTFS-Archiv anzulegen.	local
uid	Der Eigentümer der Datei, entweder numerisch oder als Name aus passwd	root
gid	Die Gruppe der Datei, entweder numerisch oder als Name aus group	root
mode	Die Zugriffsrechte	Dateien und Geräte: rw-r—r—(644) Verzeichnisse: rwxr-xr-x(755) Verknüpfungen: rwxrwxrwx(777)

flags (type=file)	Konvertierungen vor der Aufnahme ins Tar-Archiv: *utxt* Konvertierung ins Unix-Format *dtxt* Konvertierung ins Dos-Format *sh* Shell-Skript: Konvertierung ins Unix-Format, Entfernen überflüssiger Zeichen	-
name	Alternativer Name, unter dem der Eintrag ins Archiv aufgenommen wird	-
devtype (type=node)	Beschreibt den Typ des Geräts ('c' für zeichenorientierte und 'b' für blockorientierte Geräte). Muss an zweiter Stelle stehen.	-
major (type=node)	Beschreibt die so genannte "Major"-Nummer der Gerätedatei. Muss an dritter Stelle stehen.	-
minor (type=node)	Beschreibt die so genannte "Minor"-Nummer der Gerätedatei. Muss an vierter Stelle stehen.	-
linktarget (type=symlink)	Beschreibt das Ziel der symbolischen Verknüpfung. Muss an zweiter Stelle stehen.	-

Tabelle 19: Übersicht der Optionen20

Die Dateien werden kopiert, falls die genannten Bedingungen erfüllt sind und das dazugehöre OPT_Package den Wert „yes" aufweist. Welche Opt-Variablen dazugehören, beschreibt die Datei, die sich im Verzeichnis „check" befindet.[21]

9.3. Modellerweiterung

9.3.1. Analyse

Das bisherige Modell der Pakete muss nun um Paketbeschreibungsdateien erweitert werden. Folgende Spezifikationen müssen erfüllt sein, um die Struktur einer Paketbeschreibungsdatei akkurat im bisherigen objektorientierten Modell abbilden zu können:

- Ein Paket besitzt grundsätzlich einen Namen und kann eine Paketbeschreibungsdatei enthalten.
- Die Paketbeschreibungsdatei kann eine Formatversion, sowie mehrere Paketregel enthalten. Jede Zeile der Datei ist somit eine Regel, es sei denn es handelt sich um einen Kommentar. Paketregeln enthalten beliebig viele Optionen. Wobei eine Option aus einer Bezeichnung und einem Wert besteht.
- Neben den Optionen sind der jeweilige Dateipfad, sowie mindestens eine Variable und deren geforderter Wert ebenfalls in der Paketregel vereint.
- Kommentare können am Zeilenbeginn oder nach den Pfaden bzw. Optionen in den Paketbeschreibungsdateien auftauchen.

[20] Quelle: fli4l-3.9.0-rev24742, Abbildung 8.1: Optionen für Dateien
[21] Quelle:Entwickler-Dokumentation „8.5.4 Die Liste der zu kopierenden Dateien"

Das folgende Fachklassendiagramm bildet die oben beschriebenen Punkte ab.

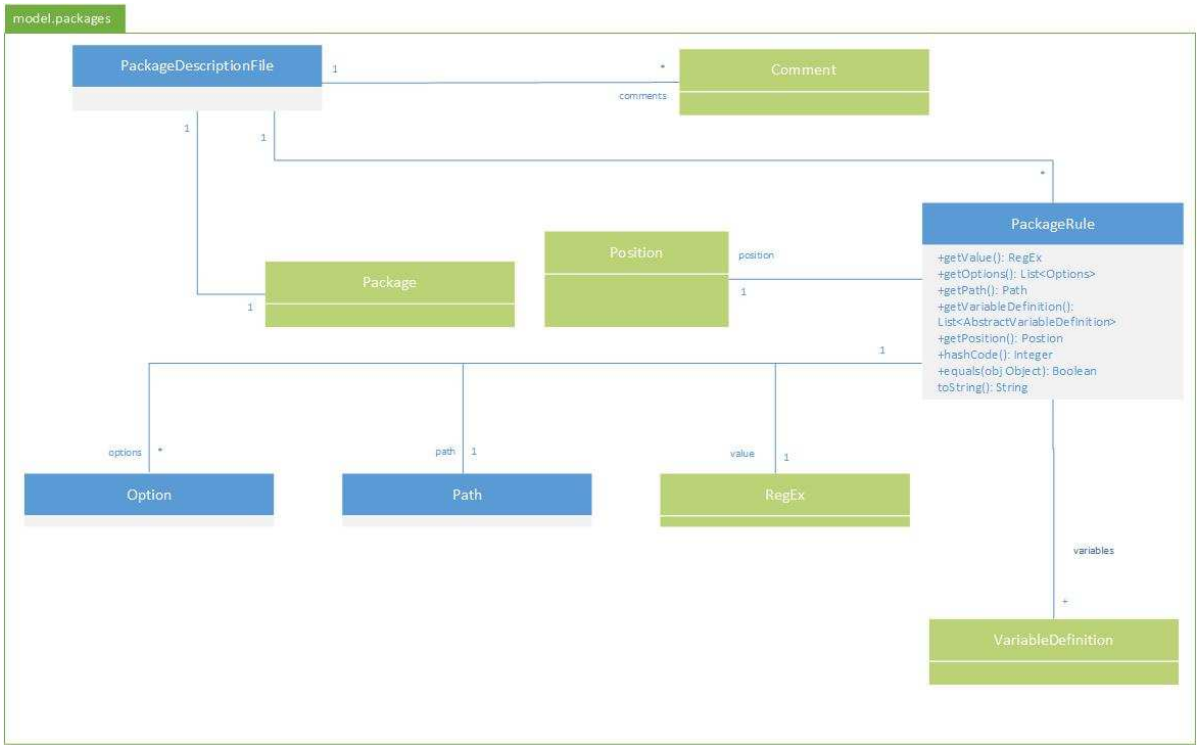


Abbildung 69: Modell – Fachklassendiagramm

Die grün hinterlegten Klassen wurden der Übersicht halber vereinfacht dargestellt.

9.3.2. Entwurf

Grundsätzlich wurde auf das bereits vorhandene Modell zurückgegriffen. Somit ist die bereits vorhandene Struktur verwendet worden. Diese musste geringfügig verändert beziehungsweise erweitert werden, um die Spezifikationen der Paketbeschreibungsdateien zu berücksichtigen.

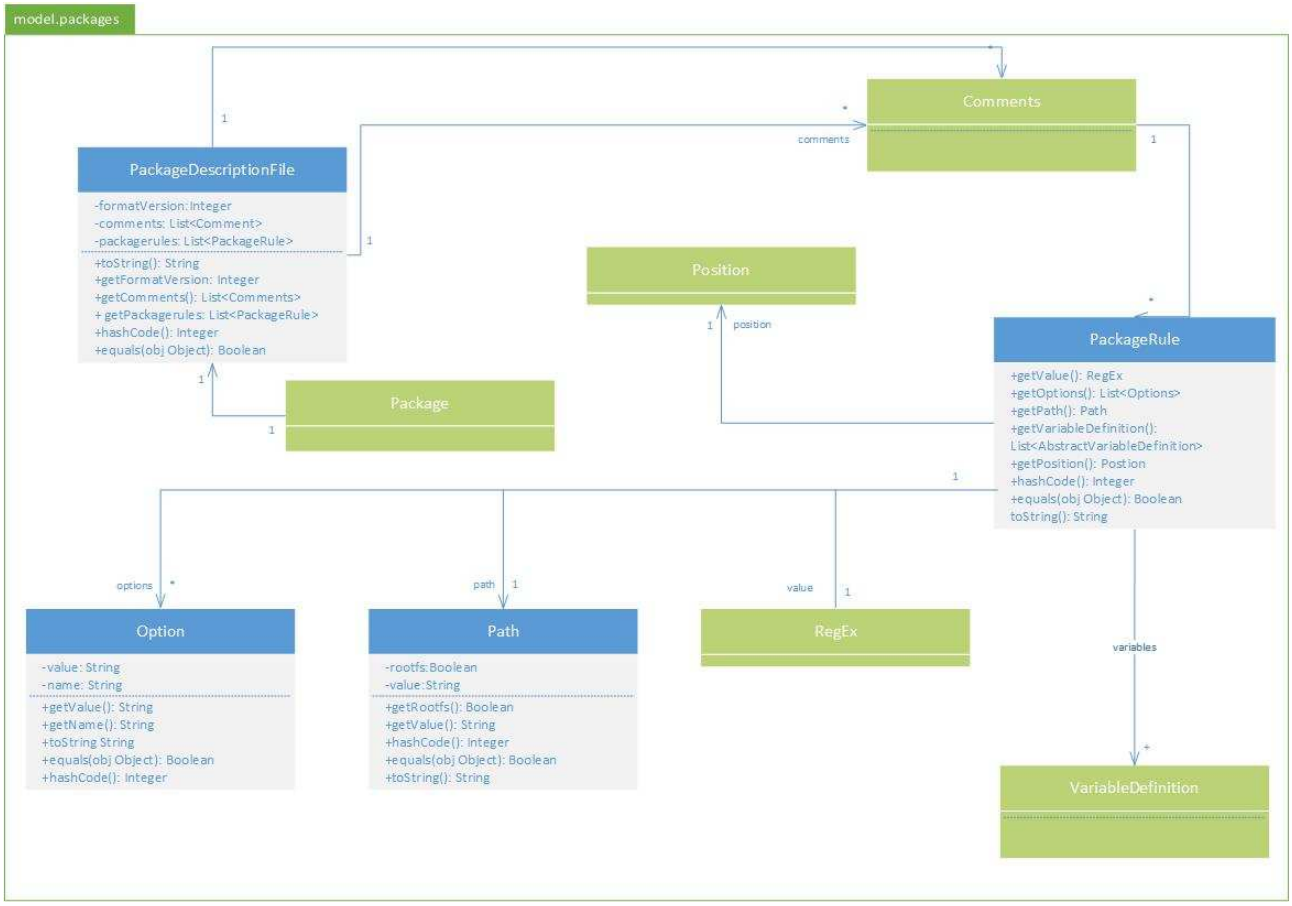


Abbildung 70: Modell - Klassendiagramm

Die folgenden Details sind bei der Betrachtung der Abbildung 64 zu beachten:

Weiterhin beinhaltet der Pfad (Klasse: Path) neben einem Wert das Attribut „rootfs“. Dieses Attribut gibt an, ob der Pfad mit „rootfs:“ beziehungsweise „rootfs:files/“ beginnt. Falls die Formatversion, welche sich in der Klasse „PackageDescriptionFile“ befindet, auf den numerischen Wert null gesetzt ist, handelt es sich um den Standardwert. In diesem Fall wurde die Formatversion innerhalb der Paketbeschreibungsdatei nicht angegeben.

9.3.3. Implementierung

Das Modell wurde wie im Entwurf beschrieben umgesetzt. Weiterhin ist zu erklären, dass der Wert der Paketregel ein regulärer Ausdruck ist. Falls keine Paketregel für die Paket-Beschreibungsdatei vorhanden ist, gelten die zugehörigen Listen (Optionen, Paketbeschreibungsdateien und Kommentare) als leer.

In den folgenden Kapiteln werden der Scanner und der Parser für die Paketbeschreibungsdateien beschrieben.

9.4. Scanner

9.4.1. Analyse

Um den Scanner auf die Spezifikationen des Scannens der Paketbeschreibungsdateien vorzubereiten, musste das Vorgehen des Scanners überdacht werden. Eine syntaktische Unterscheidung, wie es bisher der Fall war, konnte weiterhin nicht verwendet werden, da die einzelnen Bestandteile auf

syntaktischem Weg nicht unterschieden werden können. Somit muss ein spaltenweises Vorgehen des Scanners implementiert werden.

Diese neue Vorgehensweise setzt einige Veränderungen voraus, da sich die grundsätzliche Arbeitsweise verändert hat. Die Symbole, in die die eingelesenen Zeichen transformiert werden, sowie das zugehörige Modell müssen den neuen Gegebenheiten angepasst werden. Diese Veränderungen sind im folgenden Kapitel einzusehen. Das folgende Fachklassendiagramm zeigt die grobe Struktur des Scanners.

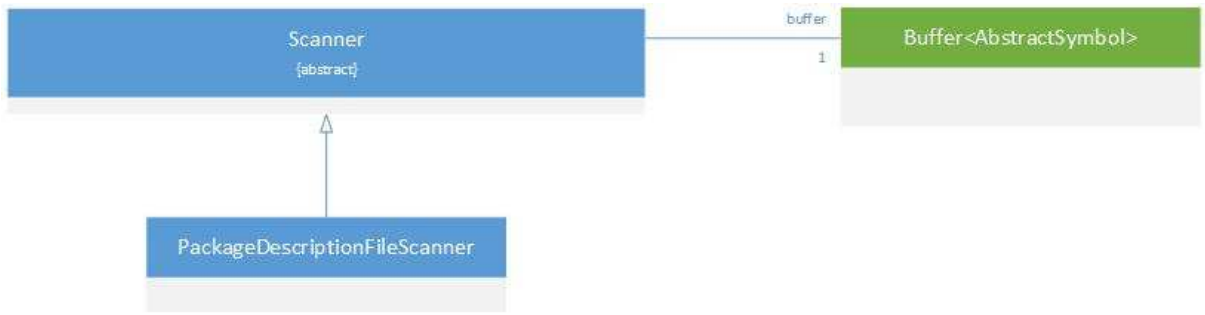


Abbildung 71: Scanner - Fachklassendiagramm

Es wird noch einen zusätzlichen Scannertyp benötigt, der Paketbeschreibungsdateiscanner, welcher wie die bisherigen Scanner einen Zustand und einen Puffer besitzt.

Diese Zustände sind in dem nachfolgenden Fachklassendiagramm dargestellt.

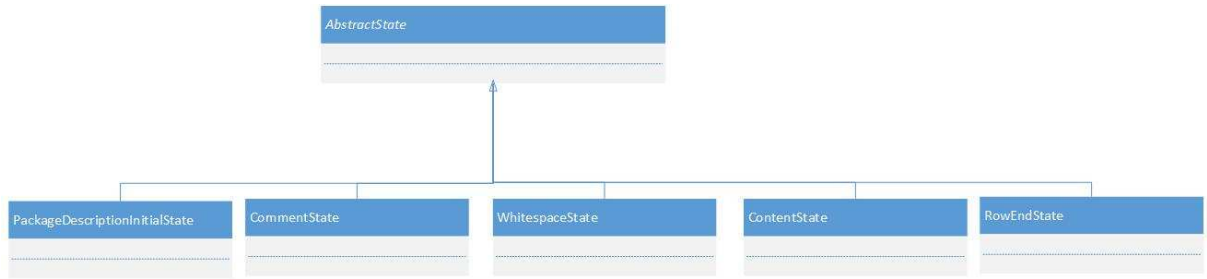


Abbildung 72: Scanner Zustände - Fachklassendiagramm

Jeder State erbt diese von dem abstrakten AbstractState. Der PackageDescriptionInitialState repräsentiert den Initialen Zustand des Scanners. Der CommentState ist der Zustand für die Kommentare. In den WhitespaceState wird bei einem Leerzeichen gewechelt und der ContentState bildet die restlichen Zeichen ab. Zu erwähnen ist noch der RowEndState der das Zeilenende wiederspiegelt. Die folgende Abbildung zeigt die Blockstruktur des Scanners.

Die Funktionsweise unterscheidet sich dahingehend vom bisherigen Scanner, dass dieser die Datei blockweise einliest. Ein Block ist immer eine Spalte dieser Datei. Es gibt demnach einen Variablen-, Wert-, Dateiblock. Jede Optionenspalte ist zusätzlich ein weiterer Block.

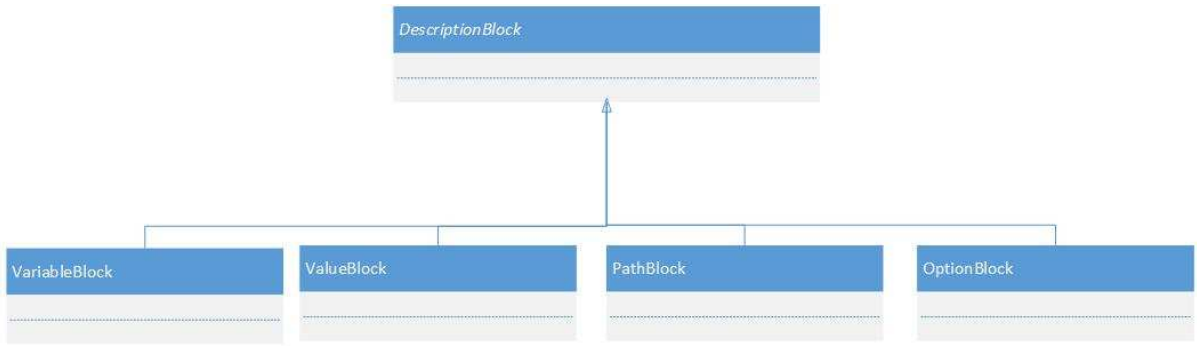


Abbildung 73: Blockstruktur - Fachklassendiagramm

Die folgende Abbildung zeigt das Fachklassendiagramm der Symbole.

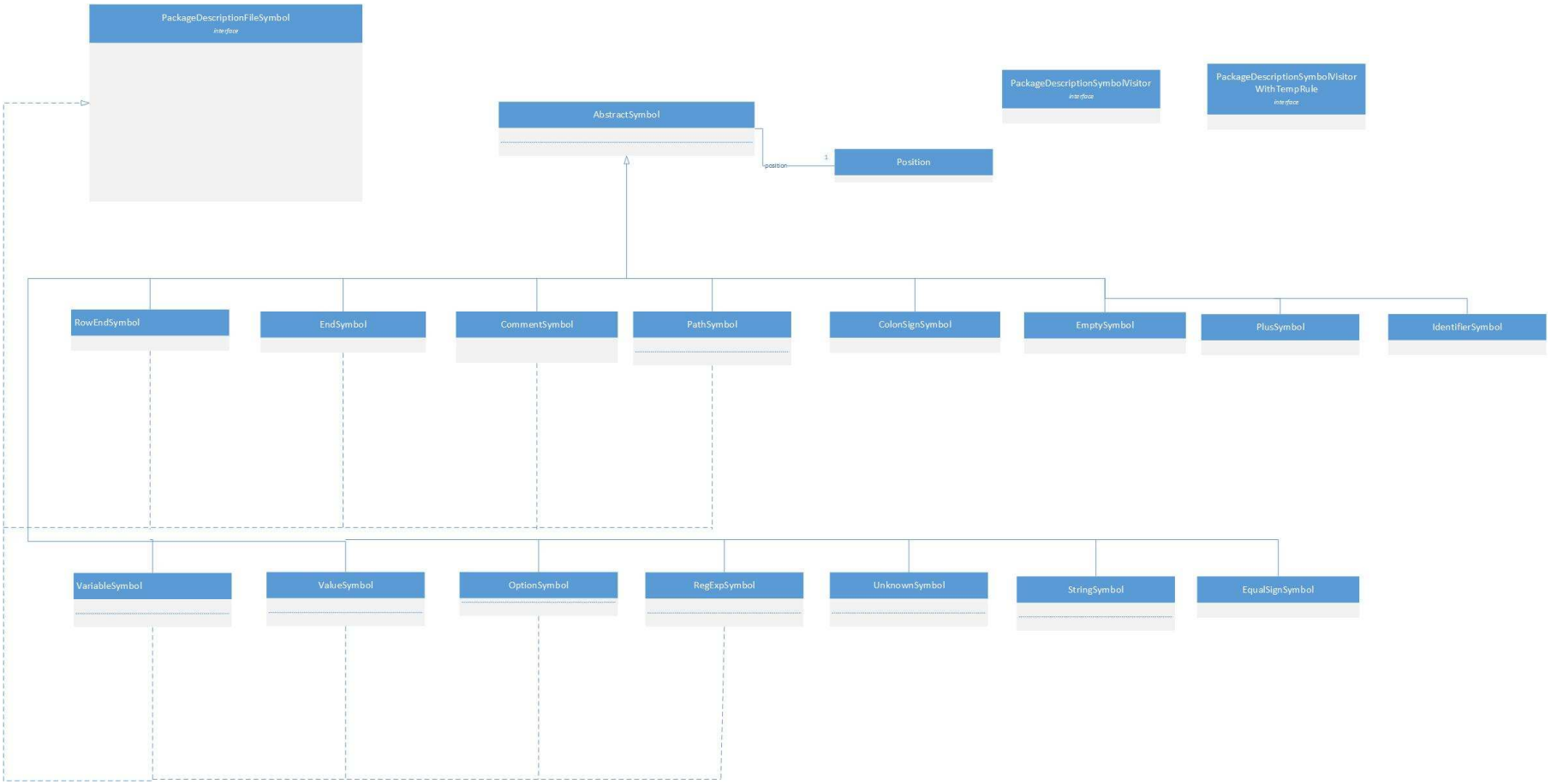


Abbildung 74: Symbole - Fachklassendiagramm

9.4.2. Aufbau der Symbole

Die folgende Grafik zeigt den Aufbau eines Kommentars.

Abbildung 75: Aufbau Kommentarsymbol

Eine Zeile mit einer Regel sieht wie folgt aus.

Abbildung 76: Aufbau Paketregel

Die oben beschriebenen Zustände lassen sich wie folgt in einem Zustandsdiagramm darstellen.

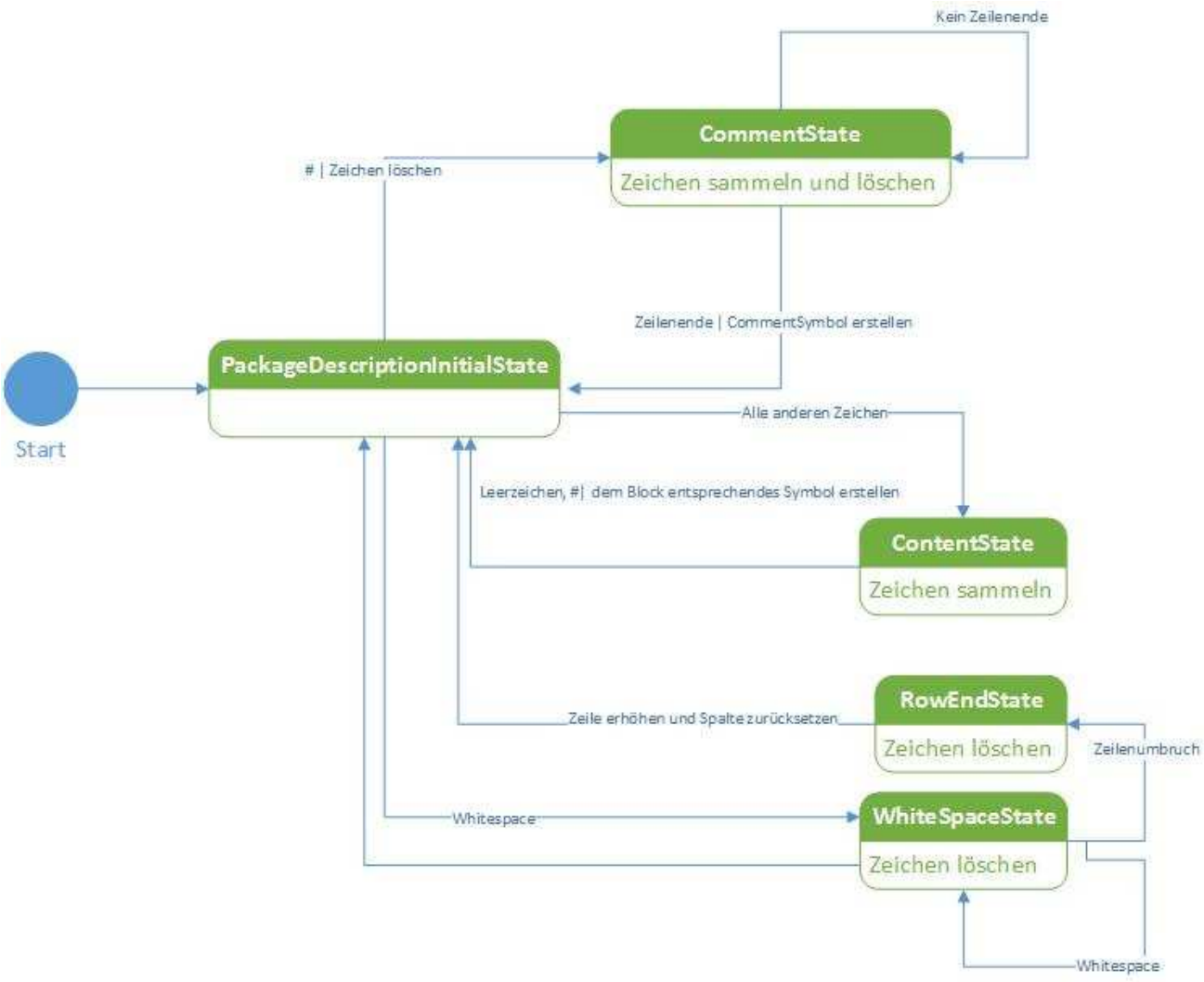


Abbildung 77: Zustandsdiagramm Scanner

Das folgende Ablaufdiagramm ist nahezu in identischer Form in der Dokumentation des Scanners vorhanden. Dies verdeutlicht die Wiederverwendbarkeit der bisherigen Scannerimplementierung.

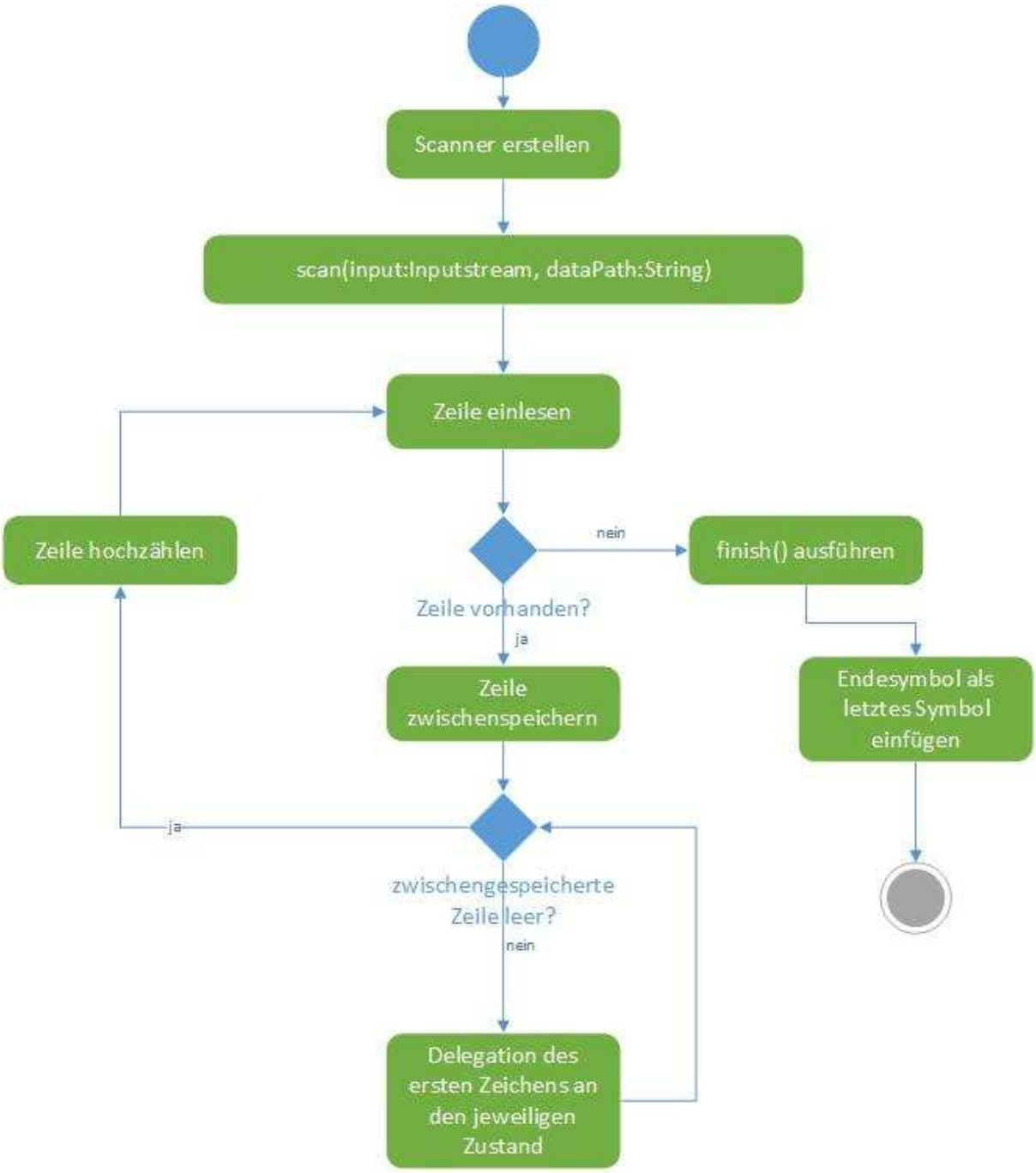


Abbildung 78: Ablaufdiagramm Scanner

Die genaue Funktionsweise des Scanners ist in Kapitel 6.4.1. zu finden. Lediglich die Abfrage, ob eine Eingabe überhaupt vorhanden ist, wurde als Vorbedingung vorausgesetzt. Dadurch fällt diese Abfrage weg. Außerdem wird nun die finish()-Methode nicht mehr nach jeder Zeile aufgerufen, sondern nur nachdem ein Endesymbol eingefügt worden ist. Der Scanner muss nun auch nicht auf den Anfangszustand zurückgesetzt werden, da für jede Datei ein neuer Scanner erzeugt wird.

Zu der Analyse lassen sich die Ablaufdiagramme der einzelnen Zustände des Scanners hinzufügen.

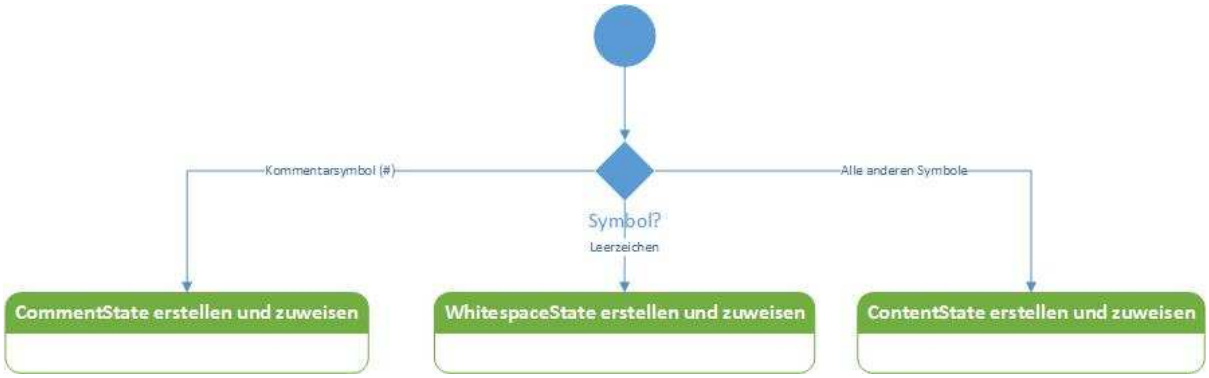


Abbildung 79: Scannerablauf PackageDescriptionInitalState

Die obere Abbildung beschreibt den Ablauf des „PackageDescriptionInitialStates“ des Scanners. Dieser entscheidet je nach Symbol, welcher Zustand erstellt und zugewiesen werden soll. Bei einem Kommentarsymbol wird der Kommentarzustand (CommentState) aufgerufen und zugewiesen. Im Fall eines oder mehrerer Leerzeichen wird ein Leerzeichenzustand (WhitespaceState) erstellt und als nächstes zugewiesen. Bei allen anderen Symbolen wird ein Inhaltszustand (ContentState) erstellt und in diesen gewechselt.

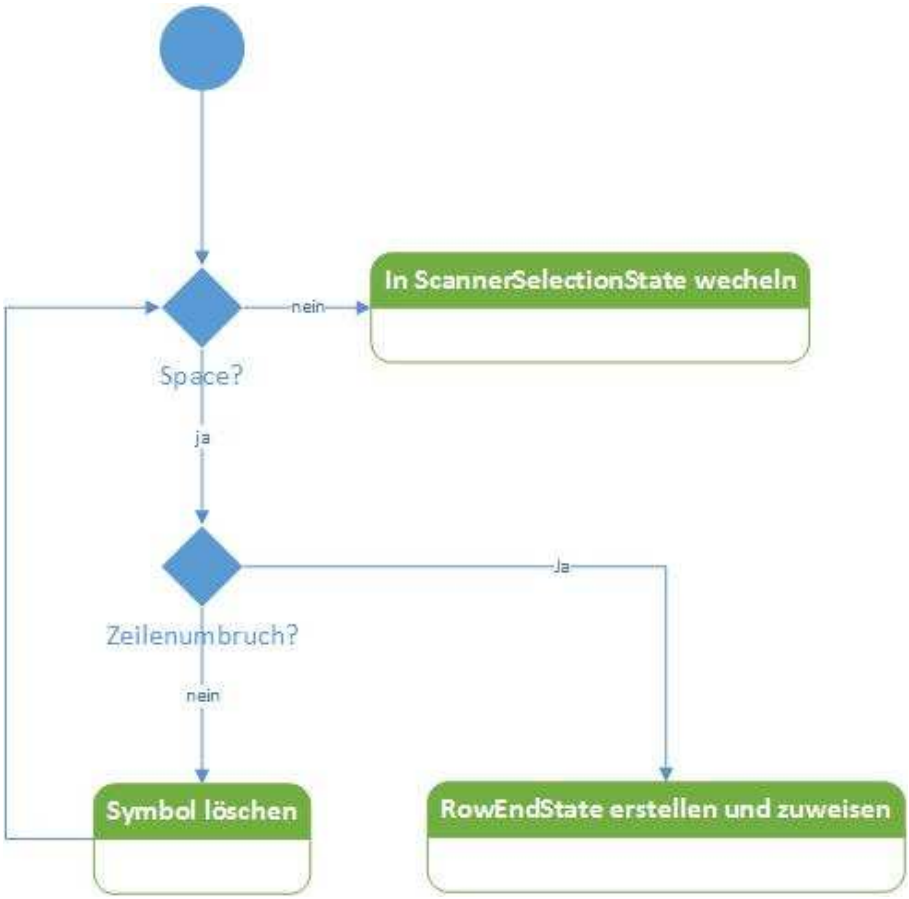


Abbildung 80: Scannerablauf WhitespaceState

In dem Leerzeichenzustand (WhiteSpaceState) wird zu Beginn geprüft, ob das Symbol ein Leerzeichen ist. In dem negativen Fall wird in den „SelectionState“ zurückgewechselt. Im positiven Fall wird überprüft, ob das Zeichen ein Zeilenumbruch ist. Dann wird es im Anschluss gelöscht. Wenn dies der Fall ist, wird in den entsprechenden Zeilenendezustand gewechselt.

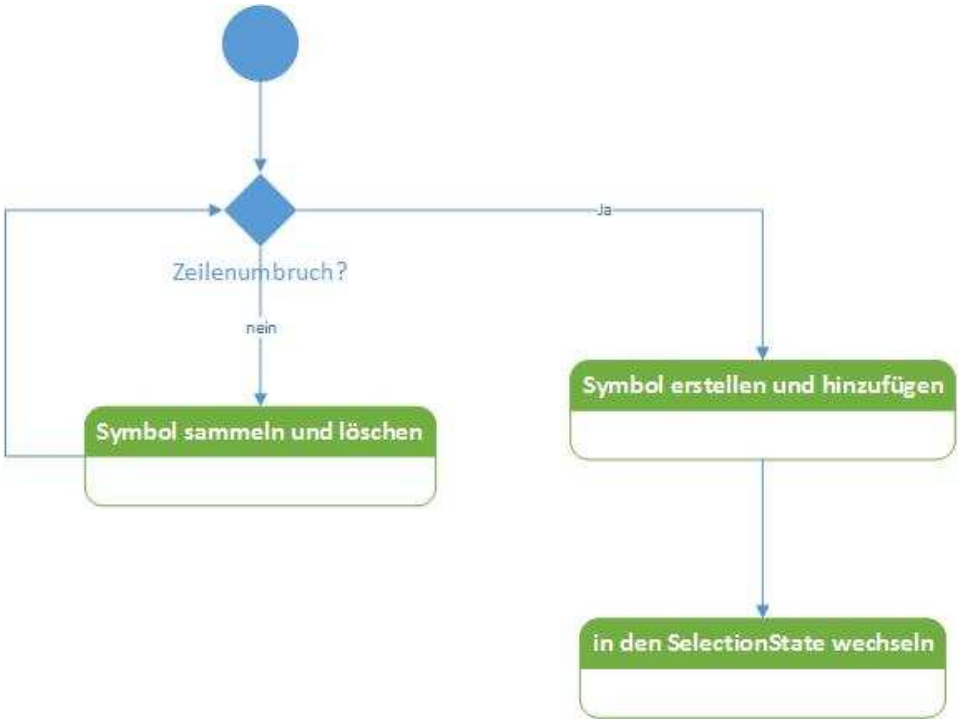


Abbildung 81: Scannerablauf CommentState

Die obere Abbildung beschreibt den Ablauf des Kommentarzustands (CommentState). Wenn das Symbol kein Zeilenumbruch ist, wird dieses gesammelt und gelöscht und wieder überprüft. Bei einem Zeilenumbruch wird das Kommentarsymbol erstellt und dieses wird dem Puffer hinzugefügt.

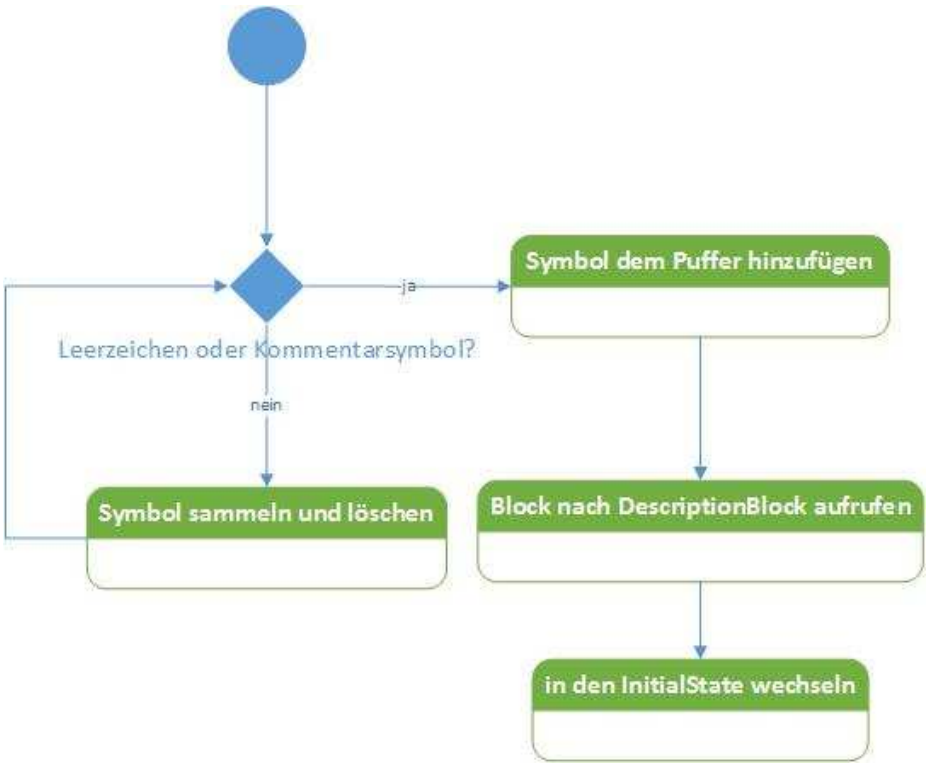


Abbildung 82: Scannerablauf ContentState

Diese Abbildung beschreibt den Ablauf des Inhaltszustands (ContentState). Bei keinem Leerzeichen wird das Symbol gesammelt und anschließend gelöscht. Kommt jedoch ein Leerzeichen oder ein

Kommentarsymbol, so wird dies Sammeln unterbrochen und das Symbol wird dem Puffer hinzugefügt. Auf dem Beschreibungsblock wird der nächste Zustand aufgerufen und in dem Scanner gespeichert.

9.4.3. Entwurf

Das Fachklassendiagramm wird nun durch das folgende Klassendiagramm konkretisiert.

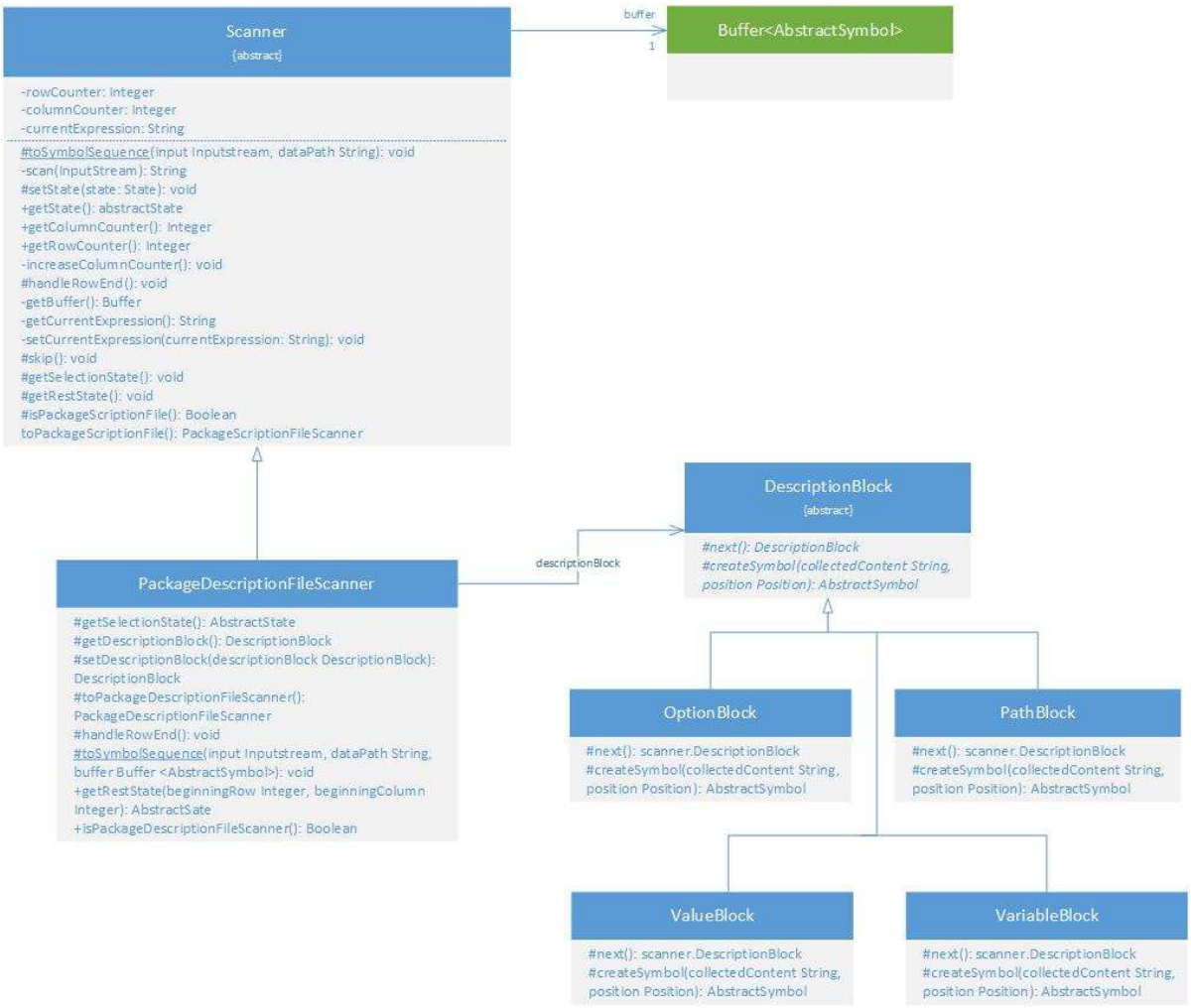


Abbildung 83: Scanner Klassendiagramm

Das Klassendiagramm des Scanners zeigt, dass es möglich war den Aufbau des Scanners in seinen groben Zügen zu belassen. Ein Scanner ist weiterhin in Besitz eines Puffers, in welchem er die gescannten Zeichen einfügen kann. Weiterhin wurde für die Transformierung der Zeichen in Symbole das Zustandsentwurfsmuster genutzt. Für die Paketdateien ist lediglich ein weiterer spezieller Scanner erforderlich. Die ursprünglichen drei speziellen Scanner finden beim Scanvorgang der Paketbeschreibungsdateien keine Verwendung. Stattdessen muss ein neuer Scanner definiert werden, der Paketbeschreibungsdateiscanner, hier vertreten durch die Klasse „PackageDescriptionFileScanner“. Ein solcher Scanner arbeitet blockweise, somit werden die verschiedenen Spalten als Blöcke eingelesen. Die Methode „next()“ ruft den nächsten Zustand auf. Die Methode „createSymbol“ bewirkt nun die Symbolerstellung

Das neue Klassendiagramm der Symbole nimmt nun folgende Form an:

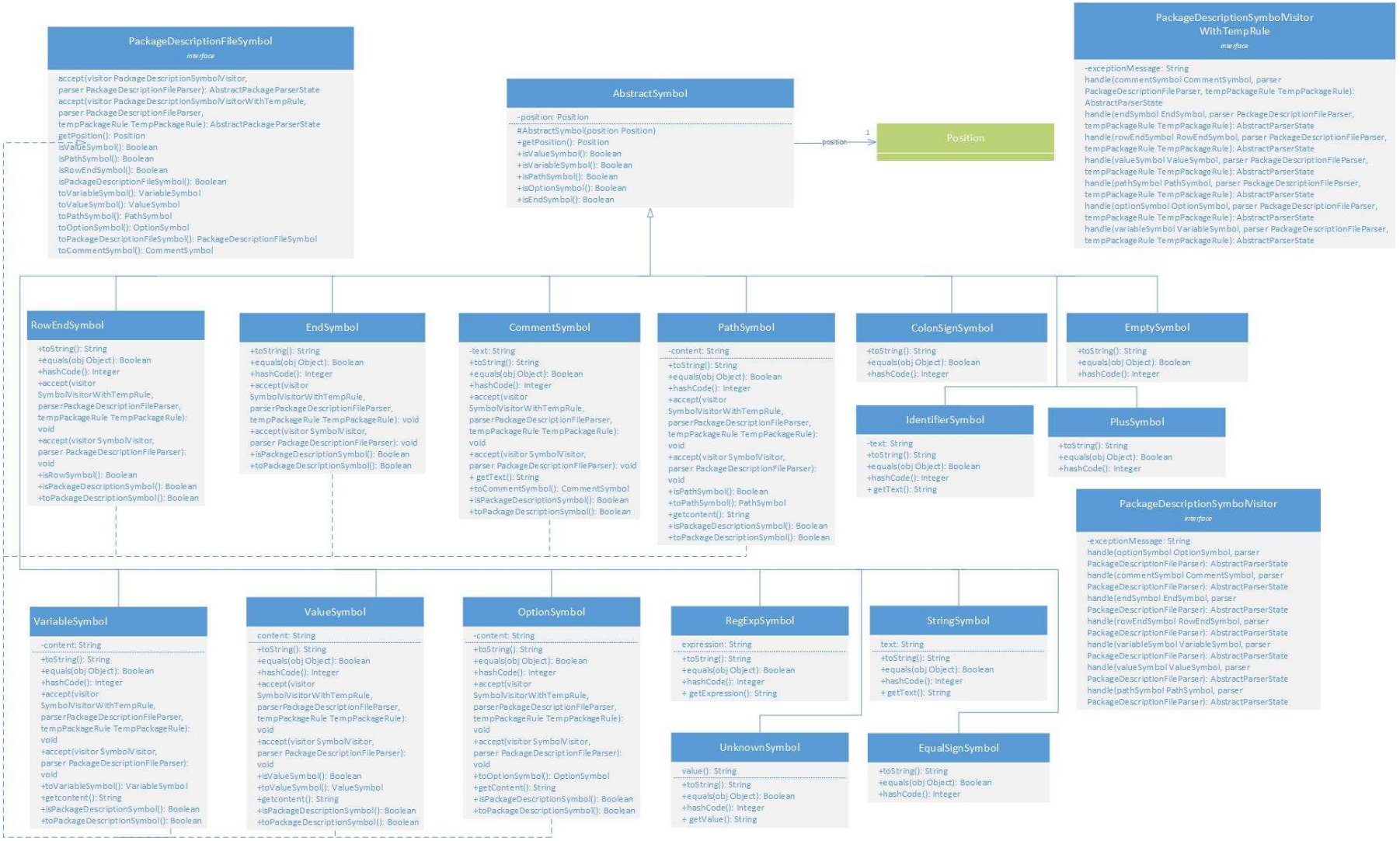


Abbildung 84: Symbol - Klassendiagramm

Die Symbole werden je nach Zeichen in unterschiedliche Klassen unterteilt Ebenfalls die Angabe einer Position eines Symbols ist weiterhin vorhanden. Hinzugekommen sind die Klassen „VariableSymbol“, „OptionSymbol“, „ValueSymbol“, „RowEndSymbol“ und „PathSymbol“. Diese Symbole vertreten die einzelnen Spalten. Besonders hervorzuheben ist das Endesymbol, hier vertreten durch die Klasse „Endsymbol“. Nur wenn ein Endesymbol vorhanden ist, terminiert der Parser. Voraussetzung ist hierbei, dass der Scanner immer ein Endesymbol liefert, sogar wenn ein Fehlerfall auftritt ist. Diese Bedingung ist von äußerster Wichtigkeit für eine richtige Funktionsweise des Parsers. Darüber hinaus wurde das Besucher-Entwurfsmuster verwendet, um von folgenden Vorteilen zu partizipieren:

- Die Besucher haben nun die Möglichkeit, über mehrere Klassenhierarchien zu agieren.
- Operationen lassen sich durch das Hinzufügen neuer Besucher definieren.
- Die Trennung von besucherfremdenden Operationen und verwandten Operationen.
- Eine zentrale Verwaltung von verwandten Operationen ist in einem Besucher möglich.

Dieses Entwurfsmuster wird durch die Schnittstellen „PackageDescriptionSymbolVisitor“ und „PackageDescriptionSymbolVisitorSymbolVisitorWithTempRule" vertreten. Diese Trennung der beiden Symbolbesucher war nötig, da der Parser an einigen Stellen eine bereits erstellte temporären Paketregeln benötigt, um weitere Informationen hinzuzufügen. An anderen Stellen, an denen ein Besucher benötigt wird, ist jedoch noch keine temporäre Paketregel vorhanden, sodass zwei unterschiedliche Besucher benötigt werden.

Zusätzlich zu neuen Klassen und deren Attribute sind den Symbolklassen neue Methoden hinzugefügt worden. Die abstrakte Klasse „AbstractSymbol“ wurde um Abfragen ergänzt, welche erkennen lassen, ob es sich bei diesem Symbol um einen Wert-, Variable-, Pfad-, Option-, Zeilenende-, Kommentar- oder Endesymbol handelt. Diese Methoden, die ebenfalls in der Schnittstelle „PackageDescriptionFileSymbol“ vorhanden sind, müssen nicht in den unteren Hierarchieklassen überschrieben werden, da eine Verbindung zwischen der Schnittstelle und der Implementierung in der Klasse „AbstractSymbol“ automatisch aufbaut wird. Diese Schnittstelle wird nur von denjenigen Symbolklassen implementiert, welche in den Paketbeschreibungsdateien vorhanden sein können. Eine weitere Methode, welche einer Erklärung bedarf, ist „accept“. Diese Methode muss implementiert sein, um der Definition des Besucherentwurfmusters gerecht zu werden. Diese ermöglicht das Annehmen eines Besuchers. Da es zwei verschiedene Besucher gibt, beinhaltet jede konkrete Klasse zwei Implementierungen dieser Methode, wovon jede auf einen der beiden Besucherspezialisiert ist. Die to<Spaltentyp>Symbol- Methoden ermöglichen eine Veränderung der Klassenzugehörigkeit (cast). Bei einer nicht möglichen Änderung wird eine Ausnahme ausgelöst. Damit wird verhindert, dass die Klasse nicht unschön direkt die Klassenzugehörigkeit ändert(cast).

9.4.4. Implementierung

Die Implementierung des Scanners ist nach dem Aufbau des Entwurfs geschehen. Die Funktionsweise des Blockzählers wird nun im Folgenden erläutert.

Hierzu musste ein Beschreibungsblock dem Zustandsmuster entsprechend eingeführt werden. Der Inhaltszustand greift in der „createSymbol“- Methode auf die Blöcke zurück und fordert diese auf, ein dem Block entsprechendes Symbole zu erstellen. Weiterhin bewegt der Inhaltszustand das vom Be-

schreibungsblock erstellte Symbol in den zugehörigen Puffer und fordert den entsprechenden Block dazu auf den nächsten Block als aktuellen Block zu setzen.

Der Scanner ist nun auf das Scannen einer Paketbeschreibungsdatei vorbereitet. Im Folgenden werden der Parser und dessen Anpassungen erläutert.

9.5. Parser

9.5.1. Analyse

Das bisher implementierte Vorgehen der Parser kann zum parsen der Paketbeschreibungsdateien größtenteils wiederverwendet werden. Der größte Unterschied besteht in der Verwendung des Besucher- Entwurfsmusters. Neben den im Kapitel Scanner – Entwurf genannten allgemein gültigen Vorteilen, war die zusätzliche Struktur, Übersichtlichkeit und Redundanzvermeidung dieses Entwurfsmusters ausschlaggebend für dessen Einführung. Das folgende Fachklassendiagramm zeigt die grobe Struktur des Parsers für die Paketbeschreibungsdateien.

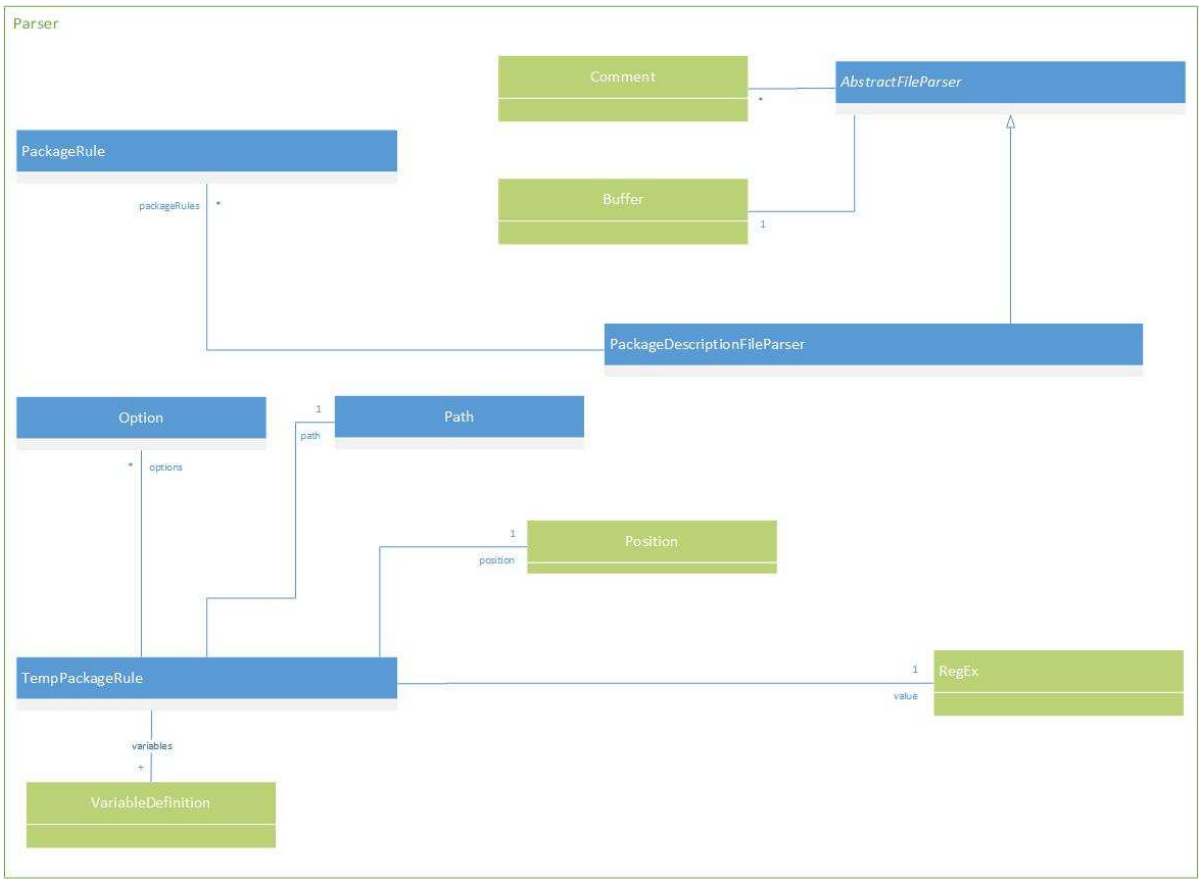


Abbildung 85: Parser - Fachklassendiagramm

Wie die bisherigen Parser hat auch dieser einen momentanen Zustand. Weiterhin arbeitet der Parser zunächst lediglich mit temporären Paketregeln, bis diese vollständig sind und sie der Paketbeschreibungsdatei hinzugefügt werden können.

Wie bereits bekannt, ist eine Paketbeschreibungsdatei aus vier Spalten aufgebaut. Jeweils eine Spalte für Variable, Wert, Datei und Optionen. Die erste Spalte, die der Variablen, besteht aus einer kommaseparierten Liste und die zweite Spalte, die Wertespalte, besteht aus einem regulären Ausdruck. Die dritte Spalte besteht aus dem Dateipfad der zu übernehmenden Datei. Die Optionen können aus

mehreren Spalten bestehen, es müssen allerdings keine Optionen vorhanden sein. Diese Spalten bestehen aus dem Namen der jeweiligen Option, einem Gleichheitszeichen und einem anschließenden Wert. Weiterhin ist es möglich, dass Dateien leer sein können, oder auch nur Kommentare enthalten können.

Das sich für die Implementierung des Parsers das Zustandsentwurfsmuster anbietet, ist eine Übersicht der möglichen Zustände nötig. Das folgende Fachklassendiagramm stellt die möglichen Zustände, die vom Parser angenommen werden können dar.

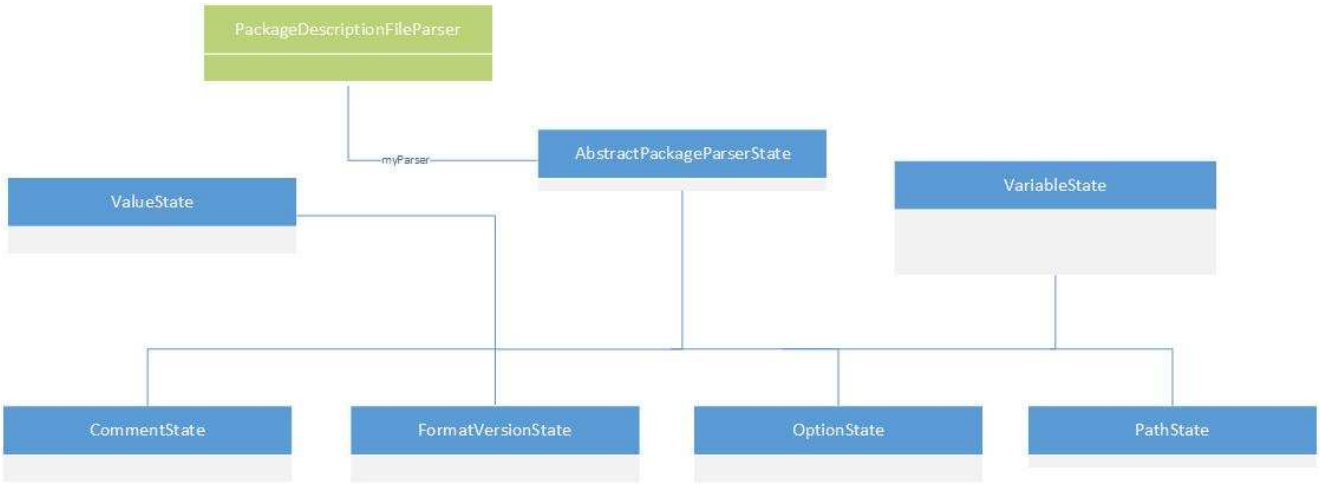


Abbildung 86: Parser Zustände - Fachklassendiagramm

Die Symbole der Paketbeschreibungsdateien haben folgende Varianten:

Sie können entweder Variablen, Werte, Kommentare, Formatversionen, Optionen oder Pfade annehmen. Für jede dieser Möglichkeiten ist ein Zustand zu implementieren.

Das folgende Ablaufdiagramm zeigt den abstrakten Ablauf des Parsers. Die einzelnen Abläufe werden nach dem Zustandsdiagramm des Parsers erläutert.

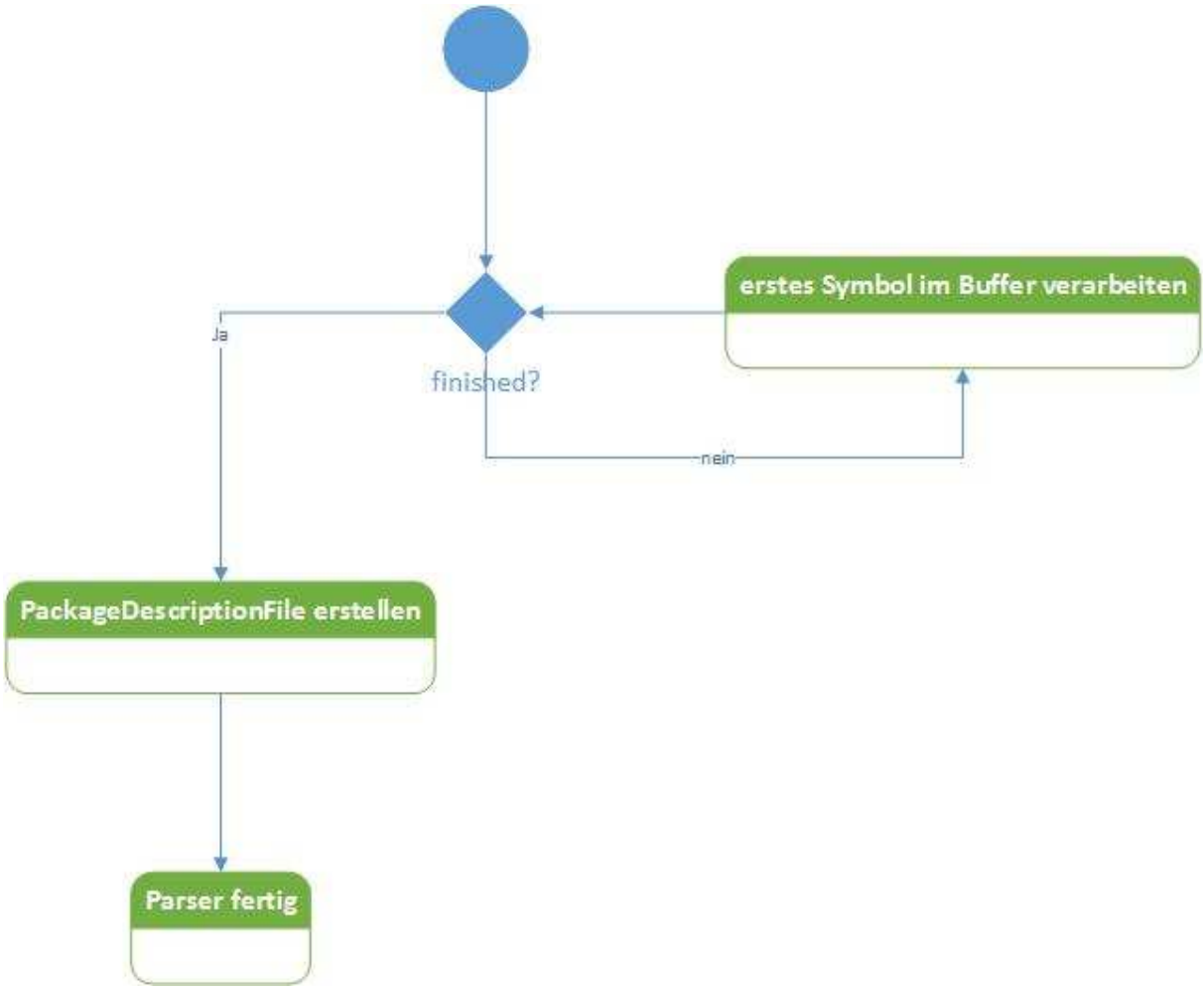


Abbildung 87: Parser - Ablaufdiagramm

Im Allgemeinen arbeitet der Parser so, dass zunächst überprüft wird, ob bereits ein Endesymbol verarbeitet wurde. Falls dem nicht so ist, wird das erste bzw. jeweils nächste Symbol aus dem Puffer verarbeitet. Nach den jeweiligen Übergängen werden Paketregeln erstellt. Wurde ein Endesymbol verarbeitet, wird ein 'PackageDescriptionFile' erstellt und der Parser ist fertig. Das folgende Diagramm zeigt die Zustände des Parsers und deren Übergänge.

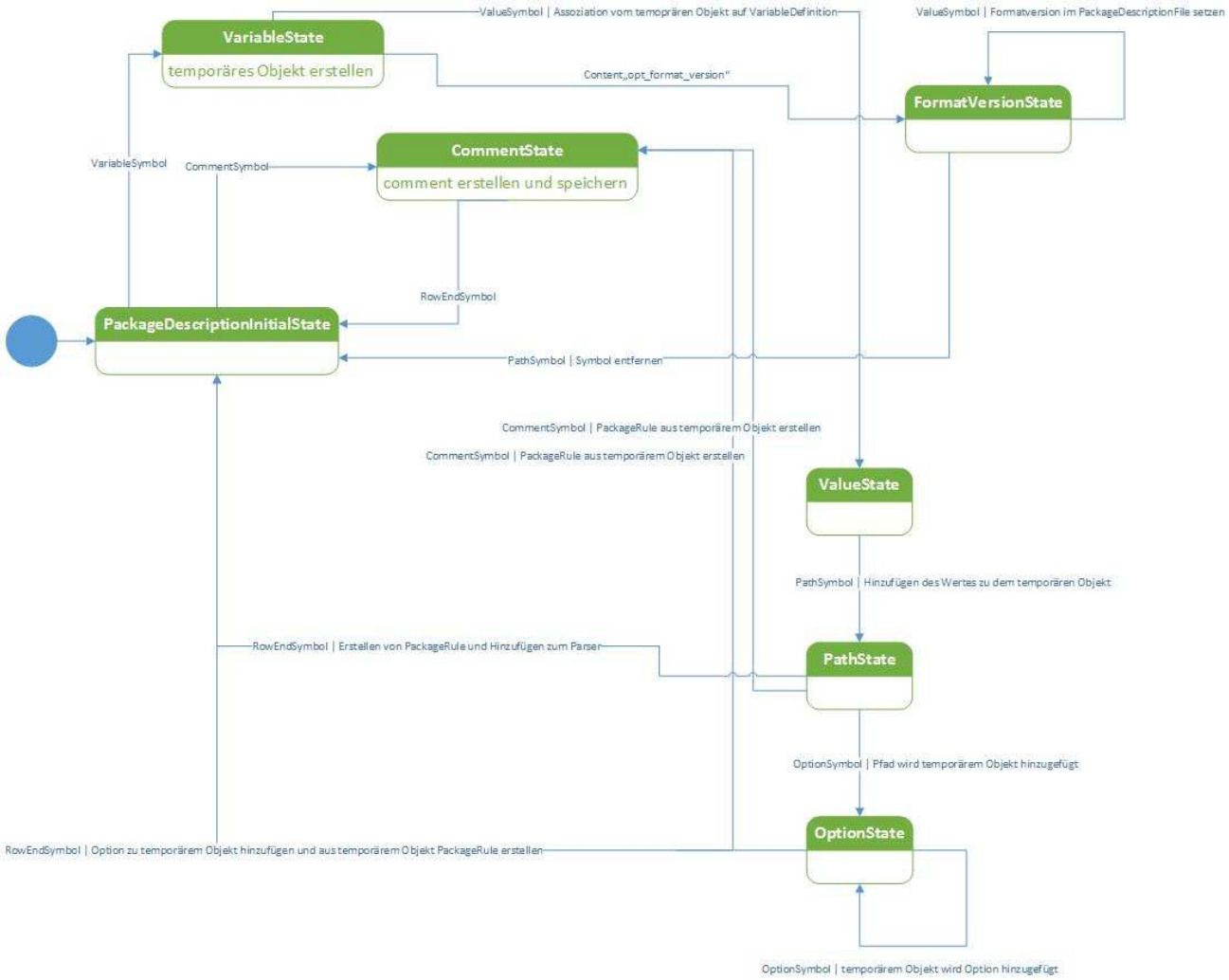

ValueSymbol | Assoziation vom temoprären Objekt auf VariableDefinition
ValueSymbol | Formatversion im PackageDescriptionFile setzen
VariableState
temporäres Objekt erstellen
Content„opt_format_version"
FormatVersionState
VariableSymbol
CommentSymbol
CommentState
comment erstellen und speichern
RowEndSymbol
PackageDescriptionInitialState
PathSymbol | Symbol entfernen
CommentSymbol | PackageRule aus temporärem Objekt erstellen
CommentSymbol | PackageRule aus temporärem Objekt erstellen
ValueState
PathSymbol | Hinzufügen des Wertes zu dem temporären Objekt
RowEndSymbol | Erstellen von PackageRule und Hinzufügen zum Parser
PathState
OptionSymbol | Pfad wird temporärem Objekt hinzugefügt
OptionState
RowEndSymbol | Option zu temporärem Objekt hinzufügen und aus temporärem Objekt PackageRule erstellen
OptionSymbol | temporärem Objekt wird Option hinzugefügt

Abbildung 88: Parser – Zustandsdiagramm

Die obere Abbildung zeigt die verschiedenen Zustände, die der Parser annehmen kann. Der Ausgangszustand ist der Initialzustand. Grundsätzlich wird nun erwartet, dass die folgenden Eingaben in den Parser der Spaltenstruktur folgen. Somit wird zunächst ein Variablensymbol erwartet, dann ein Wertesymbol, ein Pfadsymbol und Fakultativ ein Optionssymbol. Da keine Optionsblöcke kommen müssen, kann bereits das Zeilenendesymbol nach dem Pfadblock kommen. Das Einlesen eines Zeilenendesymbols bewirkt nach dem Options-, Pfad-, Kommentar- und Formatversionssymbol das Erstellen einer Packetregel und das Hinzufügen dieser zum Parser.

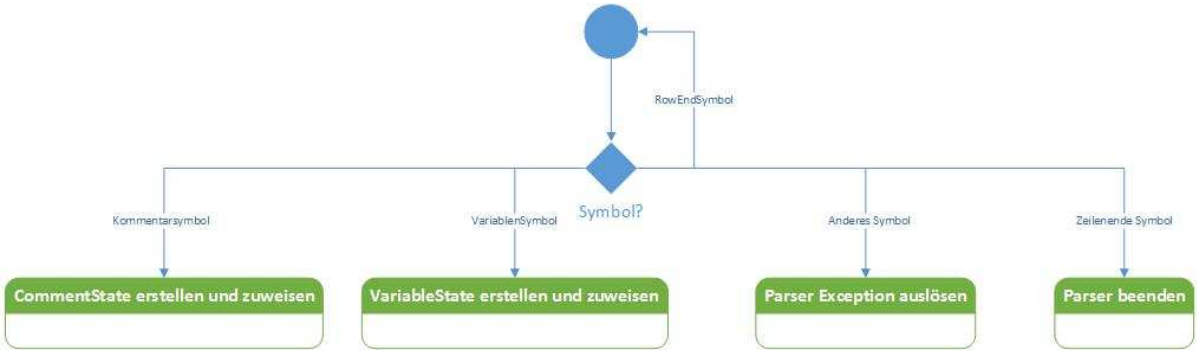


Abbildung 89: Initialsierungszustand – Ablaufdiagramm

Nachdem der Parser in den Initialzustand gewechselt ist wird auf das erste bzw. nächste Symbol im Puffer überprüft. Handelt es sich hierbei um ein Kommentarsymbol wird in den Kommentarzustand gewechselt, bei einem Variablensymbol in den Variablenzustand. Ist das kommende Symbol ein Zeilenendesymbol verbleibt der Parser im Initialzustand, bei einem Endesymbol wird der Parser beendet. Sollte ein anderes Symbole eingelesen werden, wird eine Ausnahme ausgelöst.

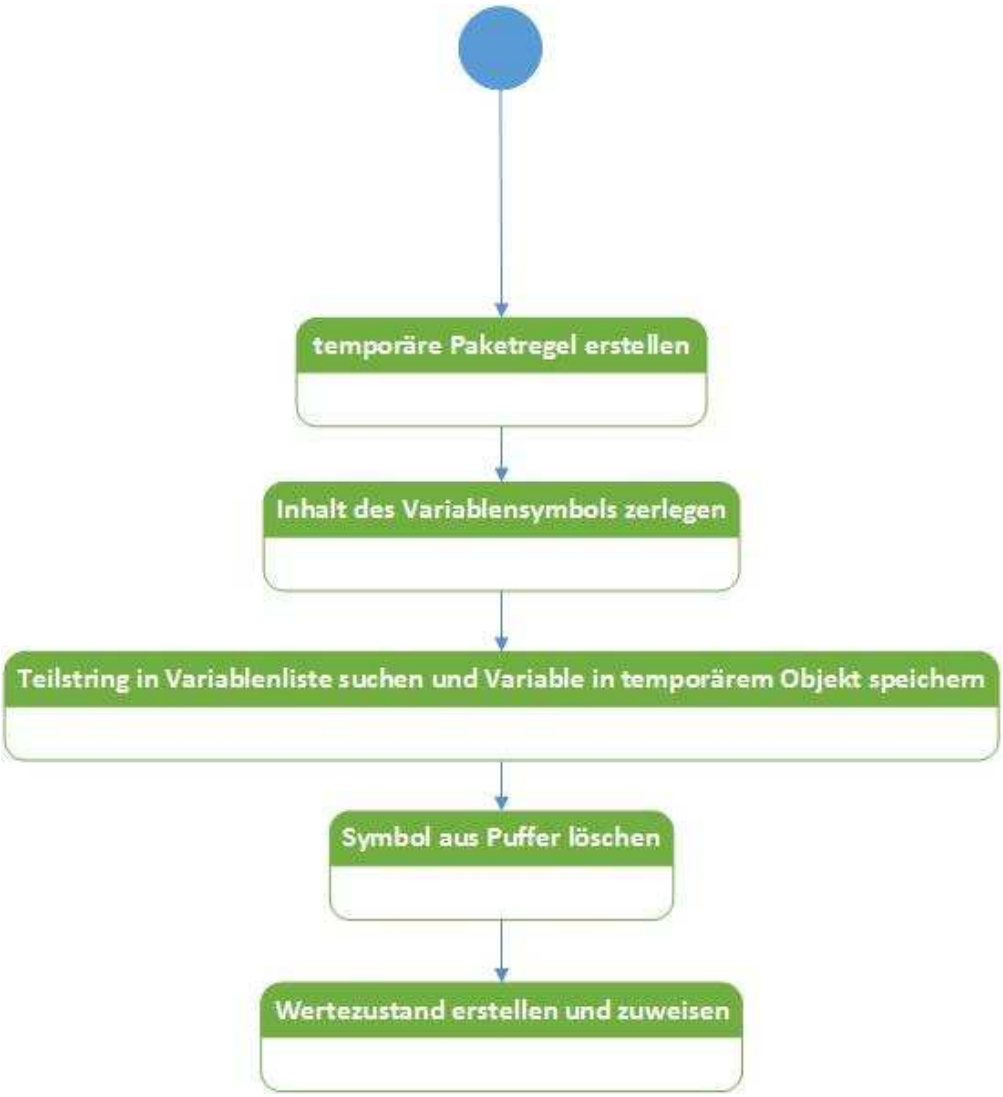

temporäre Paketregel erstellen
Inhalt des Variablensymbols zerlegen
Teilstring in Variablenliste suchen und Variable in temporärem Objekt speichern
Symbol aus Puffer löschen
Wertezustand erstellen und zuweisen

Abbildung 90: Variablenzustand - Ablaufdiagramm

Nachdem der Parser in den Variablenzustand gewechselt ist, wird zuerst eine temporäre Paketregel erstellt. Danach wird der Inhalt des Variablensymbols zerlegt (da es sich um eine Komma getrennte Liste handelt) und der jeweilige Teilstring in der bereits geparsten Variablenliste gesucht und die dazugehörige Variable im temporären Objekt gespeichert. Danach wird das eben verarbeitete Pfadsymbol aus dem Puffer entfernt und in den Wertzustand gewechselt.

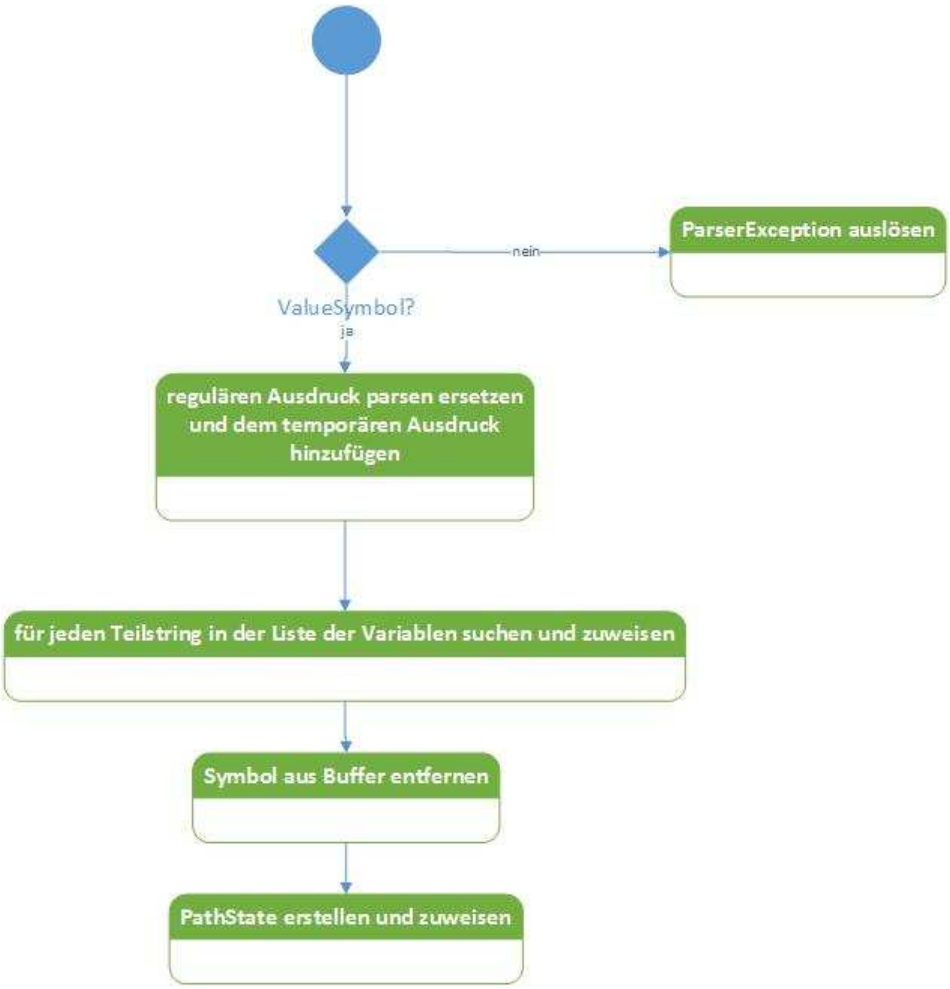

ParserException auslösen
nein
ValueSymbol?
ja
regulären Ausdruck parsen ersetzen und dem temporären Ausdruck hinzufügen
für jeden Teilstring in der Liste der Variablen suchen und zuweisen
Symbol aus Buffer entfernen
PathState erstellen und zuweisen

Abbildung 91: Wertzustand: Ablaufdiagramm

Nachdem der Parser in den Wertzustand gewechselt ist, wird geprüft, ob das kommende Symbol auch tatsächlich ein Wertsymbol ist, da zuvor direkt in diesen Zustand gewechselt wurde. Falls dem nicht so ist, wird eine Ausnahme ausgelöst. Ansonsten wird das Wertsymbol weiterverarbeitet, indem der Inhalt des Wertsymbols in einen regulären Ausdruck transformiert (sofern dies möglich ist) und letztendlich im temporären Objekt gespeichert wird. Danach wird das eben verarbeitete Wertsymbol aus dem Puffer entfernt und in den Pfadzustand gewechselt.

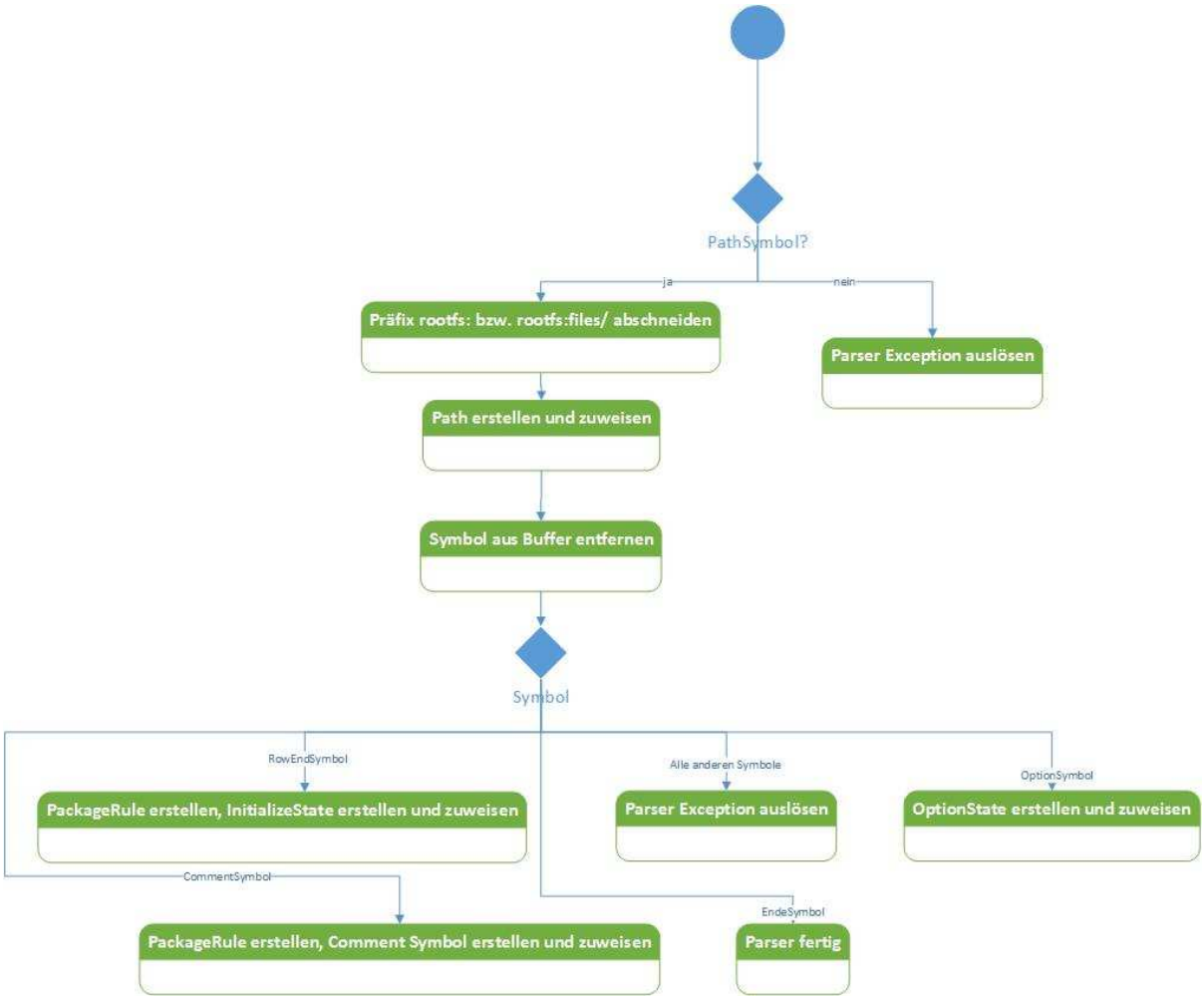


Abbildung 92: Pfadzustand – Ablaufdiagramm

Nachdem der Parser in den Pfadzustand gewechselt ist wird geprüft, ob das kommende Symbol auch tatsächlich ein Pfadsymbol ist, da zuvor direkt in diesen Zustand gewechselt wurde. Falls dem nicht so ist, wird eine Ausnahme ausgelöst. Ansonsten wird das Pfadsymbol weiterverarbeitet, indem zuerst ein evtl. vorhandenes Präfix („rootfs:“ bzw. „rootfs:files/“) abgeschnitten wird und der verbliebene Rest des Pfadsymbols in ein Pfadobjekt transformiert und der temporären Paketregel hinzugefügt. Danach wird das eben verarbeitete Pfadsymbol aus dem Puffer entfernt. Es erfolgt noch eine Überprüfung auf das nächste Symbol. Bei einem Optionssymbol wird in den Optionszustand gewechselt. Bei einem Kommentarsymbol wird eine Paketregel erstellt und in den Kommentarzustand gewechselt. Bei einem Zeilenendesymbol wird analog zum Kommentarsymbol verfahren, wobei hier in den Initialzustand gewechselt wird. Bei einem Endesymbol ist der Parser fertig. Sollte keines dieser Symbole zutreffen wird eine Ausnahme ausgelöst.

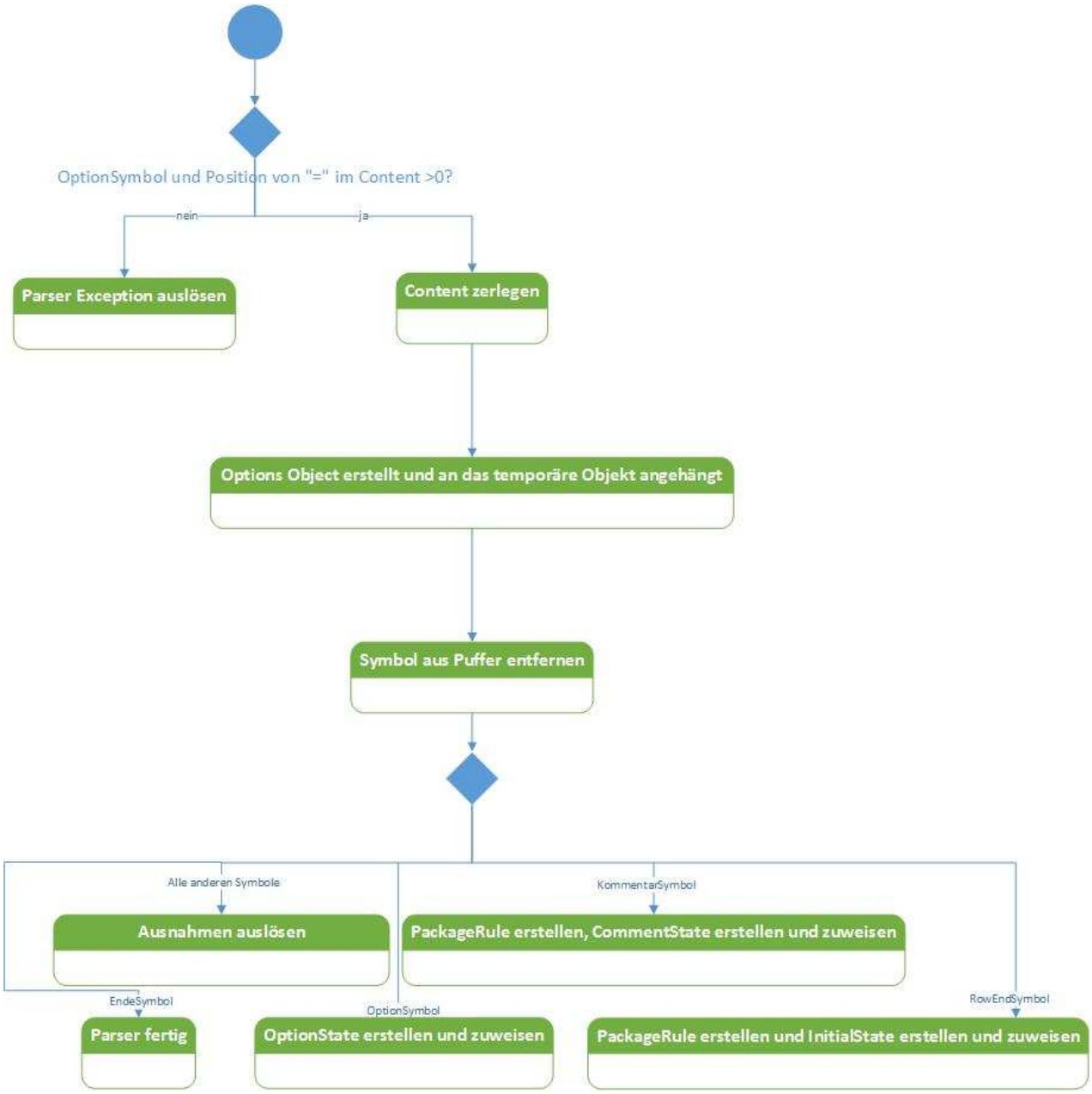


Abbildung 93: Optionszustand – Ablaufdiagramm

Nachdem der Parser in den Optionszustand gewechselt ist, wird geprüft, ob das kommende Optionssymbol (z.B.“name=wert“) ein Gleichheitszeichen enthält und dieses nicht an erster Stelle ist. Falls dem nicht so ist, wird eine Ausnahme ausgelöst. Ansonsten wird der Inhalt des Optionssymbols zerlegt, sowie ein Optionsobjekt erstellt und an das temporäre Objekt angehängt. Danach wird das eben verarbeitete Optionssymbol aus dem Puffer entfernt. Es erfolgt noch eine Überprüfung auf das nächste Symbol. Bei einem Optionssymbol wird erneut in den Optionszustand gewechselt. Bei einem Kommentarsymbol wird eine Paketregel erstellt und in den Kommentarzustand gewechselt. Bei einem Zeilenendesymbol wird analog zum Kommentarsymbol verfahren, wobei hier in den Initialzustand gewechselt wird. Bei einem Endesymbol ist der Parser fertig. Sollte keines dieser Symbole zutreffen wird eine Ausnahme ausgelöst.

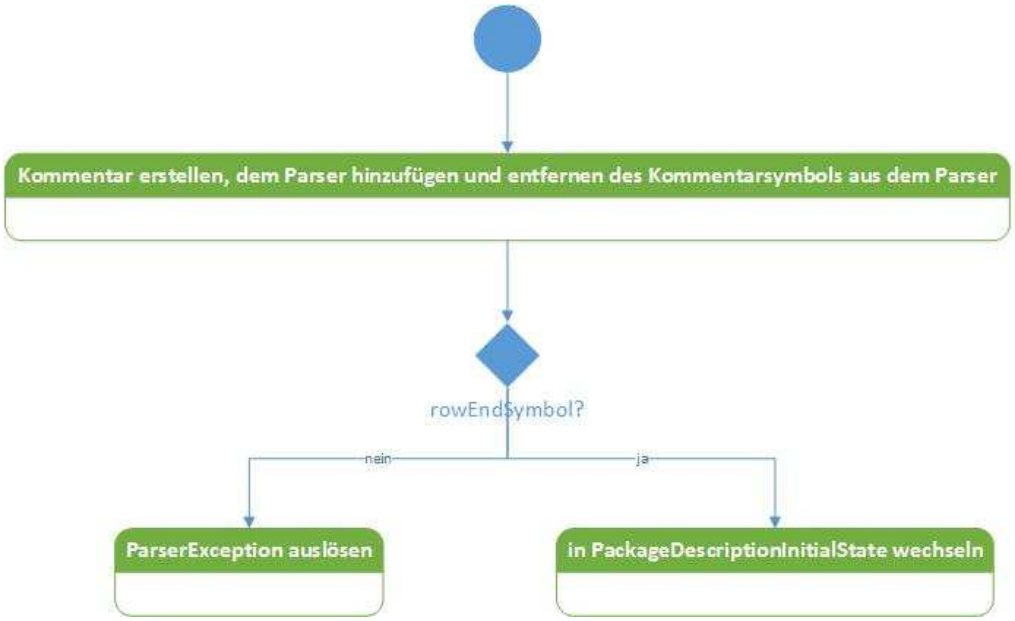


Abbildung 94: Kommentarzustand - Ablaufdiagramm

Nachdem der Parser in den Kommentarzustand gewechselt ist, wird das jeweilige Kommentarsymbol in einen Kommentar transformiert und zu der Kommentarliste des Parsers hinzugefügt. Außerdem wir das Kommentarsymbol aus dem Puffer entfernt. Im Anschluss wird ein Zeilenendesymbol erwartet. Falls es nicht vorhanden sein sollte, wird eine Ausnahme ausgelöst. Ist es vorhanden, wechselt der Parser in den Initialzustand.

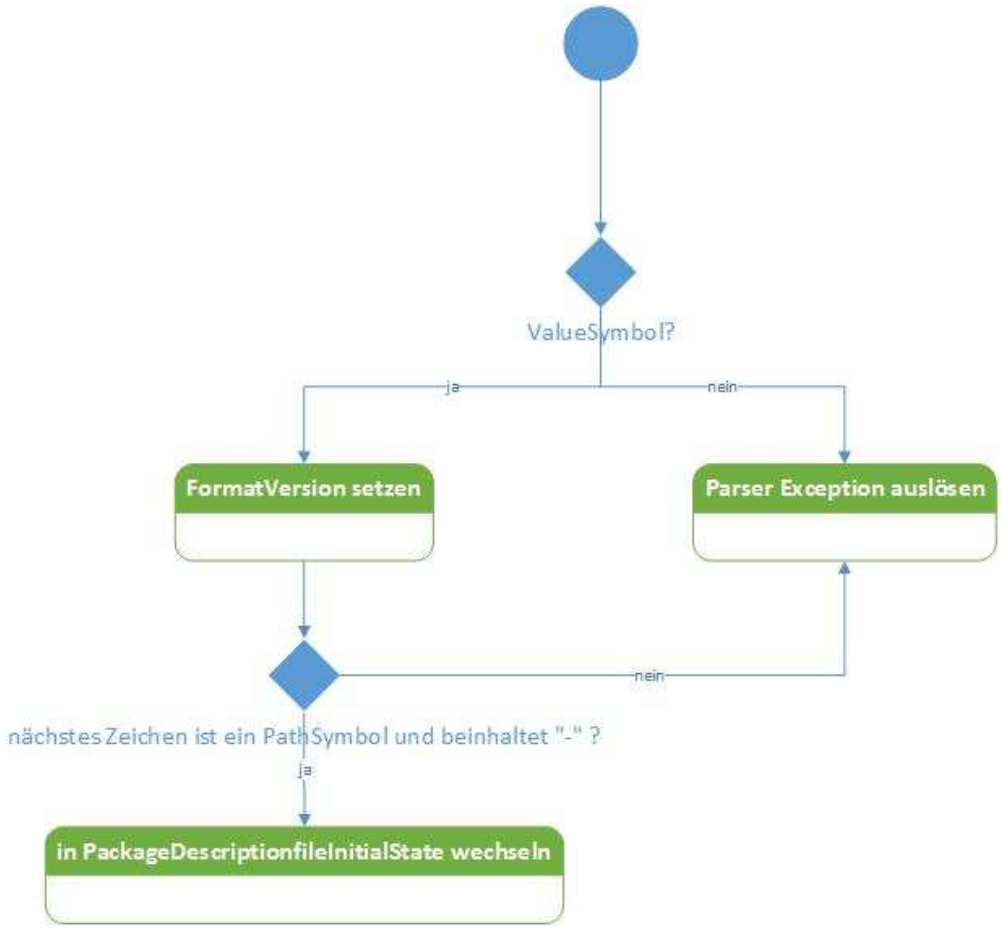


Abbildung 95: Formatversionszustand – Ablaufdiagramm

Handelt es sich zu Beginn um einen Wertsymbol, wird die Formatversion gesetzt und das nächste Symbol eingelesen. Wenn das nächste Symbol kein Pathsymbol ist, welches einen Bindestrich enthält, wird automatisch eine Ausnahme ausgelöst. Ist der Bindestrich jedoch vorhanden, wird ein Zustandswechsel in den Initialisierungszustand bewirkt. Folgt dieser Bindestrich nicht, so wird wiederum eine Ausnahme ausgelöst.

Damit sei die Analyse abgeschlossen. Das folgende Kapitel zeigt den konkreten Entwurf zur Implementierung des Parsers. Diese basiert auf der zugehörigen Analyse.

Der folgende Abschnitt zeigt den Entwurf der Parser Implementierung basierend auf der vorhandenen Analyse.

9.5.2. Entwurf

Das folgende Klassendiagramm zeigt den Aufbau des Parsers.

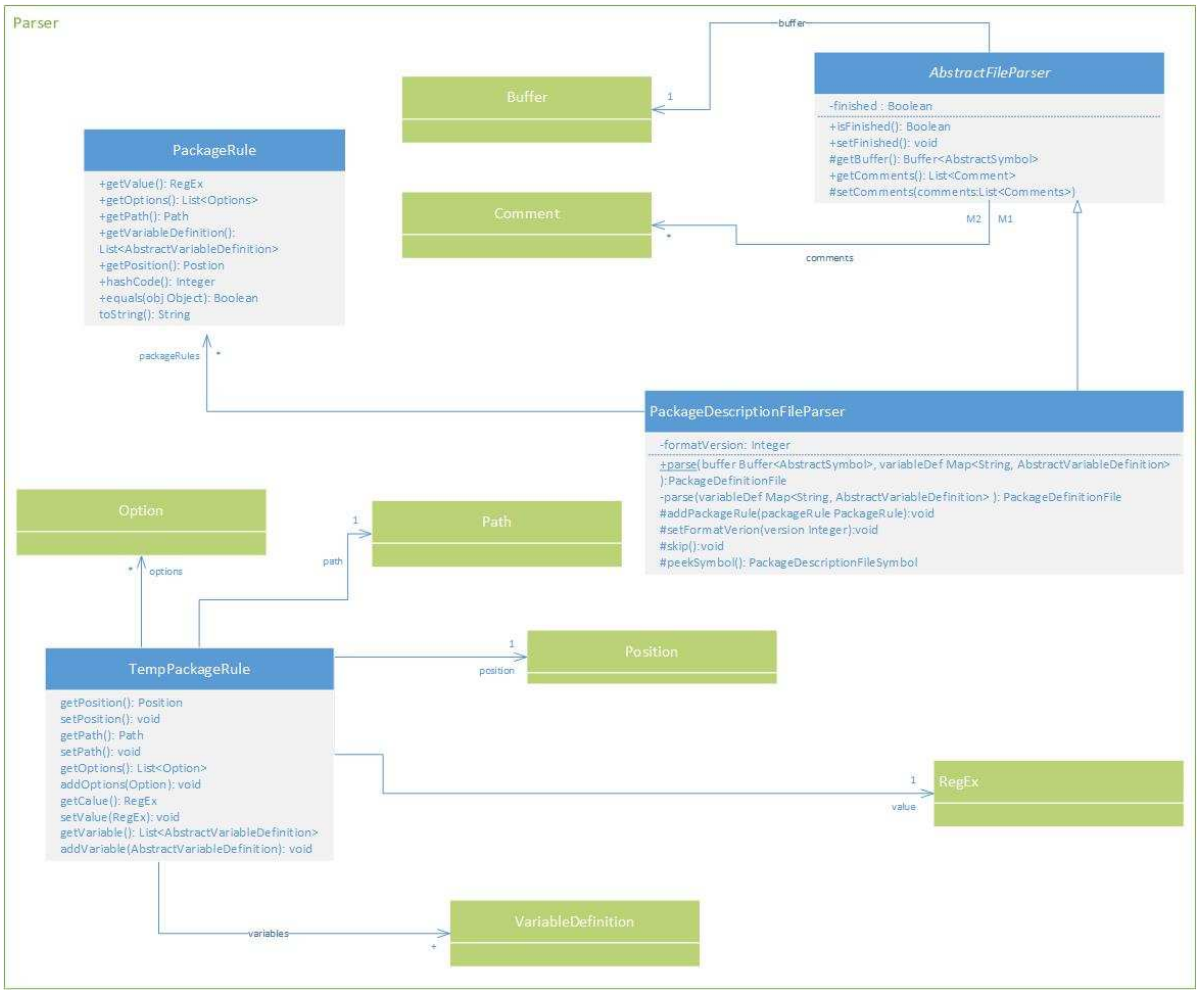


Abbildung 96: Parser - Klassendiagramm

Die obere Abbildung zeigt das Klassendiagramm basierend auf dem Fachklassendiagramm der Analyse. Die temporäre Paketregel, mit welcher der Parser arbeitet, enthält wie bereits in der Analyse erwähnt eine Variablendefinition, eine Option, ein regulären Ausdruck, eine Position und einen Pfad, ähnlich der Paketregel des Modells. Die Assoziation zwischen der Paketregel und den Blocktypen wurde hier aus Übersichtsgründen nicht aufgeführt und ist in Abbildung 5 einsehbar. Die abstrakte Klasse ‚AbstractFileParser' enthält zwei objektwertige Attribute, einen Puffer und einen Kommentar. Die grüne Färbung einer Klasse bedeutet, dass diese nicht komplett abgebildet wird, da diese bereits in anderen Abbildungen vorhanden ist. Neben dem Fachklassendiagramm des Parser ist ebenfalls das Fachklassendiagramm der Zustände des Parsers in der Analyse enthalten. Dieses wird in der folgenden Abbildung durch ein Klassendiagramm konkretisiert.

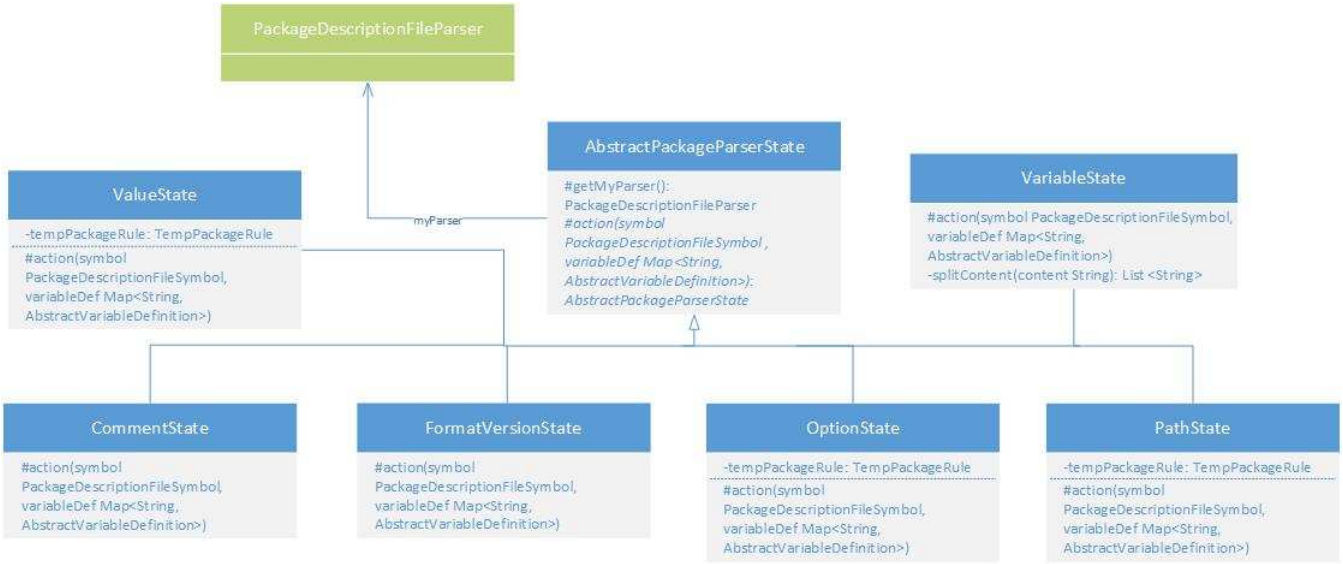


Abbildung 97: Parserzustände – Klassendiagramm

Jeder Zustand enthält eine „action- Methode“, welche die Aktion, das Parsen des jeweiligen Symbols, bei dem gegebenen Zustand auslöst. Der Variablenzustand besitzt zusätzlich die Methode „splitContent(content String)“, welche benötigt wird, da diese Spalte in kommaseparierter Form vorliegt. Die Klasse „PackageDescriptionFile“ ist hier in grüner Färbung dargestellt, da diese bereits im Klassenmodell des Parsers ausführlich behandelt wird.

9.5.3. Implementierung

Folgende Punkte wurden während der Implementierung durchgeführt und bedürfen einer gesonderten Ausführung:

- Das Zerlegen von Variablen geschieht durch die „split“- Methode des Wertzustandes. Die eigentliche Zerlegung geschieht anhand der vorhandenen Kommata.
- Das Zerlegen der Optionen geschieht im Optionszustand anhand des Gleichheitszeichens.
- Die Namen der Optionen werden nicht auf deren Bekanntheit geprüft.
- Die Variablendefinitionen werden entgegen geprüft. Dies geschieht mithilfe der Methode „searchvariabledefinition()“ der Klasse „GlobalFunctions“. Als Suchbereich werden die Variablendefinitionen des Zuweisungsdateiparsers verwendet. Der Parser verwendet das Besucherentwurfsmuster, da dieses die Möglichkeit bietet, zu erkennen welches Symbol als nächstes vorhanden ist. In diesem Punkt unterscheidet sich der Paketbeschreibungsdateiparser von der bisherigen Parsern.
- Wenn in der ersten Zeile keine Formatversion definiert wird, so wird vom Standardwert null ausgegangen.

Der Parser wurde wie im Entwurf beschrieben implementiert und ist in der Lage, Paketbeschreibungsdateien zu parsen.

9.6. Zusammenfassung

9.6.1. Abschlussfazit Implementierung

Der Scanner und der Parser wurden wie in den Entwürfen beschrieben umgesetzt. Die Testfälle überprüfen die Funktionalität der einzelnen Komponenten, sowie komponentenübergreifend.

Anzumerken ist allerdings, dass das Interface bei den Symbolen bisher nur für den „packageDescriptionFileSymbols“ umgesetzt wurde. Im weiteren Verlauf des gesamten Projektes sollte dies allerdings auch für die anderen drei Dateitypen noch ergänzt werden.

Weiterhin ist zu erklären, dass die Versionsspalte bisher nicht als weitere Überprüfung dient, sie wird somit ignoriert.

Das Besucher-Entwurfsmuster hätte auch mit einer Besucherklasse implementiert werden können, dies hätte allerdings bewirkt, dass das Attribut „tempRule“ dann in jeder zugehören Klasse vorhanden wäre. Um dies zu verhindern wurden zwei separate Besucherklassen erstellt.

Ein weiterer Punkt der im weiteren Verlauf ergänzt werden muss, sind die is- und to-Methoden. Diese existieren bisher nur für ausgewählte Symbole, sollen aber zeitnah auch in den verbleibenden Symbolen ergänzt werden.

9.6.2. Ergebnisse der Komponenten

Die Testabdeckung, welche durch eCobertura überprüft wurde, liegt bei nahezu 100%. Somit wird fast der komplette Quelltext des Scanners, Parsers und des Paketbeschreibungsmodells durch die Testfälle abgedeckt.

Die fehlenden Prozente, die zu einer vollständigen Testabdeckung fehlen, sind unter anderem auf die nicht zu testenden Konstruktoren in den Klassen der Konstanten zurückzuführen.

PMD gibt null Warnungen, wodurch die Effizienz des Codes maximiert werden konnte.

Ein einheitlicher Programmierstil ist ebenfalls vorhanden, da Checkstyle die Einhaltung dieses Stils überprüft und mit null Checkstylewarnungen dokumentiert. Programmierfehler sind laut FindBugs nicht vorhanden, diese Aussage wird durch das fehlerfreie Durchlaufen aller Testfälle bestätigt.

9.6.3. Testfälle

Um die Testabdeckung auf ein Niveau von 100% zu bringen, war eine Vielzahl an Tests nötig. Demnach ist eine große Anzahl an Testfällen vorhanden. Die Tests überprüfen die Funktionalität der einzelnen Komponenten, sowie komponenten-übergreifend. Einzelne Tests nutzen demnach auch mehrere Komponenten. Generell ist jedoch zu beachten, dass einige Testfälle auskommentiert wurden, da diese private Methoden testen. Diese Testfälle können selbstverständlich ausgeführt werden, jedoch muss vorher die getestete Methode sichtbar gemacht werden.

9.6.4. Fazit

Die Paketbeschreibungsdateien können nach dieser Implementierung gescannt und geparst werden. Weiterhin wurde die Struktur der Paketbeschreibungsdateien erfolgreich in einem objektorientierten Modell abgebildet.

10. Grafische Benutzeroberfläche - GUI

10.1. Einführung

Dieses Kapitel des Handbuchs beschäftigt sich mit dem Entwurf und der Entwicklung der grafischen Benutzeroberfläche, über die der Anwender die Konfigurationsdateien einlesen und bearbeiten kann. Wie die genaue Bedienung der Oberfläche funktioniert, sei im Handbuch der Benutzeroberfläche nachzulesen.

10.2. Analyse

Die Aufgabe der ersten fli4l-GUI (GUI = **G**raphical **U**ser **I**nterface) ist es, die Konfigurationsdateien einzulesen, diese dann anzuzeigen, zu editieren und im Anschluss speichern zu können. Bei der Erstellung der GUI soll berücksichtigt werden, dass man mehrere Router für den Anzeigebereich verwalten kann. Dafür soll für jeden Router ein eigenes Installationsverzeichnis mit den entsprechenden Konfigurationen vorliegen.

Das Editieren bezieht sich vorerst nur auf das Assignmentfile. Es ermöglicht in der ersten GUI-Version die Anpassung der Zuweisungen. Es ist nicht möglich neue Zuweisungen anzulegen oder bestehende zu löschen. Bei allen anderen Dateien soll die Möglichkeit Änderungen durchzuführen nicht erlaubt sein. Daraus folgt, dass das Speichern ebenfalls nur für das Assignmentfile von Nöten ist. Bei dem Speichern handelt es sich lediglich um eine „einfache“ Speichermöglichkeit. Bei dem der Anwender nur die Möglichkeit hat das schon vorhandene Assignmentfile durch das geänderte zu ersetzen, aber nicht die Möglichkeit bekommt einen anderen Speicherort für die geänderte Datei auszuwählen.

Zum Einlesen der Elemente in die GUI reicht es zunächst dem Anwender eine Möglichkeit zu bieten, die einzelnen Verzeichnisse schnell und einfach einlesen lassen zu können. Hierfür bietet sich ein Dialogfenster mit „Name“ und „Pfad zum Installationsverzeichnis“ an. Der „Name“ wäre der Name mit dem der Router an der Oberfläche angezeigt wird. Für die Zukunft wäre auch das persistente ablegen der eingelesenen Elemente denkbar. Dann würde der Anwender nach einem Neustart der Anwendung die schon eingelesenen Elemente vorfinden, ohne diese erneut einlesen zu müssen. Mehrere Informationen dazu befinden sich im letzten Kapitel Ausblick.

Für das Einlesen der einzelnen Dateien sind die vorhandenen Implementierungen Scanner und Parser zuständig. Das Speichern der Dateien übernimmt der schon implementierte Printer. Es muss lediglich eine Schnittstelle für das bisherige Model erstellt werden, die die bereits bestehenden Funktionen anbietet. Bevor eine Datei jedoch gescannt und geparst werden kann, muss sie aus dem Dateisystem ausgelesen werden. Dementsprechend ist zusätzlich ein „Leser“ („Reader“) für die Dateien von Nöten, der diese aus dem Dateisystem einliest und in einem geeigneten Format zur Weiterverarbeitung (Scanner/Parser) zur Verfügung stellt.

Für das Testen einer grafischen Oberfläche kann man entweder auf spezielle Tools zum Testen von Oberflächen zurückgreifen oder aber Funktionen schreiben: Die Funktionen beschreiben was der Anwender durchführen soll und welches Ergebnis erwartet wird. Zum Testen müssten alle verfassten Funktionen mindestens einmal durchgeführt werden. Bei Änderungen müssen alle Funktionen erneut durchgeführt werden. In dem hier dokumentierten GUI Projekt wurden Funktionen zum Testen der grafischen Oberfläche verwendet. (Die einzelnen Funktionen sind unter dem Kapitel „Funktionen“ aufgeführt)

Da dieser Teil der GUI nur ein erster Ausschnitt des Gesamtprogrammes darstellt, muss sehr stark auf die Erweiterbarkeit geachtet werden.

10.2.1. Aufbau einer Funktion

EineFunktion besteht aus vier Komponenten: Durchnummerierung und Name der Funktion, Hauptszenario, Häufigkeit und mögliches Fehlverhalten. Der Name beschreibt kurz welche Komponenten durch dieseFunktion getestet werden und nummeriert die Funktion. Im „Hauptszenario“ wird beschrieben, welche Aktionen der Anwender durchführen soll und welche Ergebnisse durch die ausgeführte Aktion erwartet werden. Die letzte Komponente einer Funktion ist die „Häufigkeit“. Hier wird objektiv beurteilt wie häufig eine Funktion von den Anwendern benutzt wird. Für die Einschätzung der „Häufigkeit“ wurde eine fünfstufige Skalar entwickelt: „Sehr selten“, „selten“, „normal“, „häufig“ und „sehr häufig“. Die Einschätzung wie häufig eine Funktion benutzt wird hat Einfluss auf die Gestaltung der Oberfläche und ist daher ein wichtiger Aspekt. Der Einfluss der Häufigkeit auf die Gestaltung der GUI sieht wie folgt aus:

sehr selten	Die Funktion ist ein schönes Gimmick, wird aber so selten gebraucht, dass man sie nicht auf Anhieb finden muss.
selten	Die Funktion wird für die Funktionen des Programms gebraucht, die Nutzung findet aber nur selten statt.
normal	Die Funktion wird durchschnittlich gebraucht. Sollte daher leicht zu finden sein, muss dem Anwender aber nicht gleich ins Auge fallen.
häufig	Die Funktion ist so wichtig, dass der Anwender sie leicht finden sollte. Man kann entweder einen eigenen Button dafür vorsehen oder sie in das Kontextmenü der Menüleiste unterbringe.
sehr häufig	Die Funktion wird sehr häufig gebraucht und sollte deswegen gleich auf Anhieb zu finden sein. Bei globalen Funktionen (nicht kontextabhängig) werden sie mit einem Button dargestellt.

Beim möglichen Fehlverhalten werden Fehlerfälle aufgelistet, die durch nutzer- oder systembedingtes Fehlverhalten entstehen können.

10.2.2. Toolauswahl

Als Hilfsmittel zur Entwicklung von grafischen Oberflächen kann das Tool Windowbuilder verwendet werden. Beim Windowbuilder handelt es sich um einPlug-in für das Programmierwerkzeug „Eclipse“. Mit Hilfe des Windowbuilders kann der Oberflächendesigner per Drag’n’Drop eine Oberfläche gestalten und bekommt den zugehörigen Programmcode generiert. Daher ist der Windowbuilder eine einfache Möglichkeit auch ohne Vorkenntnisse eine ansehnliche Oberfläche zu gestalten und den Entwickler von Standard-Aufgaben zu entlasten. Problematisch am Windowbuilder ist, dass der generierte Programmcode auch unnötige oder umständliche Codeabschnitte enthält und die Generierung im Großen und Ganzen auch zu einem sehr unstrukturierten Programmecode führt.

Installation über:
http://download.eclipse.org/windowbuilder/WB/integration/4.2/

Vorgezogenes Fazit: Der Windowbuilder eignet sich um erste Anfänge zu realisieren und die Zusammenhänge zwischen GUI-Elementen zu erkennen. Sobald es allerdings zu komplexen Konstellationen kommt, sollte auf die Generierung verzichtet werden. Die Nachbearbeitung des generierten Codes dauert im Zweifel länger als es direkt selbst zu schreiben.

Die folgenden beiden Abbildungen zeigen die ersten Ideen der Analysephase, die allerdings wie im nächsten Kapitel Entwurf beschrieben, noch verändert worden sind.

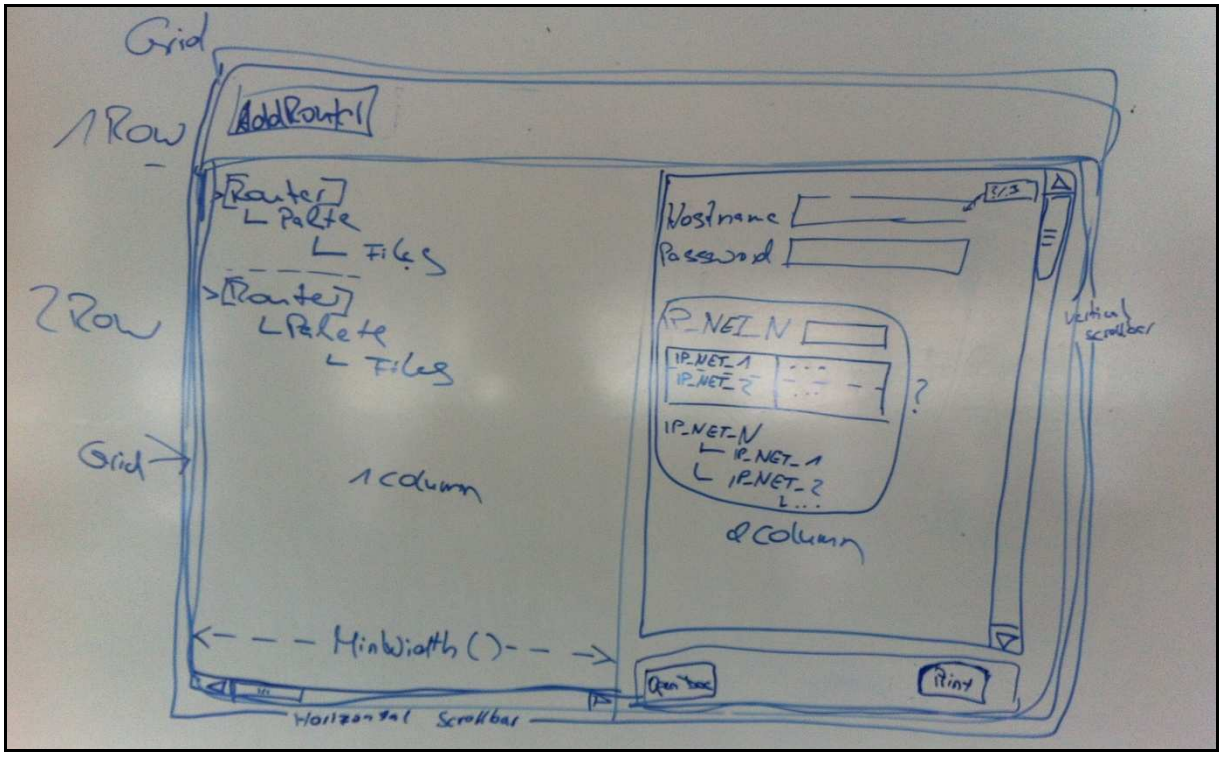


Abbildung 98: Entwurf der MainView

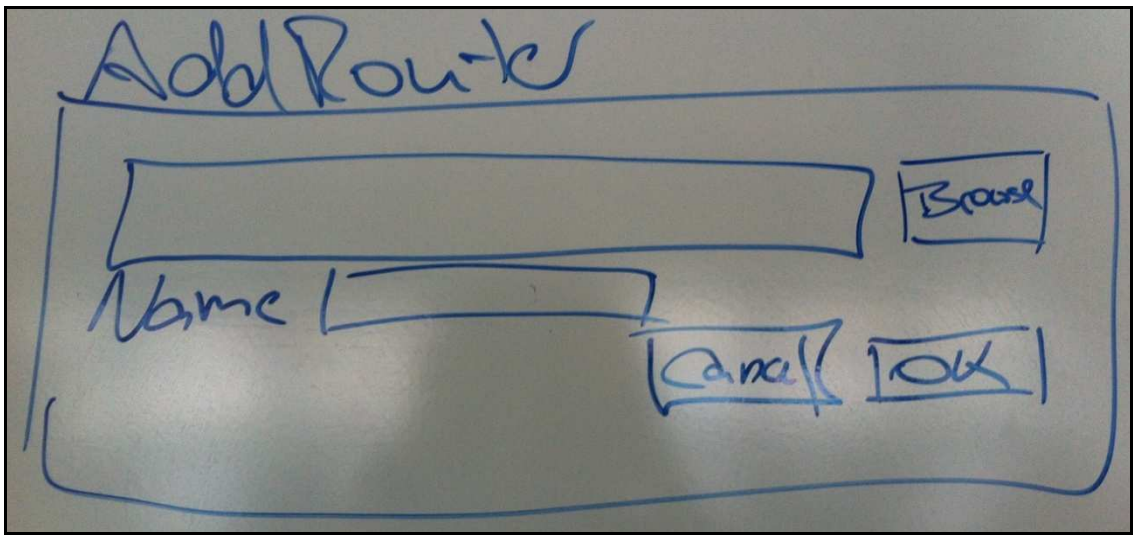


Abbildung 99: Entwurf der AddRouterView

10.3. Entwurf

10.3.1. Architektur

Wichtig an der Umsetzung der ersten GUI-Oberfläche ist, dass die Oberfläche selbst unabhängig von der Logik ist und daher zukünftig einfach ausgetauscht bzw. erweitert werden kann, ohne Teile der Logik berühren zu müssen. Aus diesem Grund bietet sich für die Umsetzung das MVC-Pattern an. MVC steht für Model-View-Controller und zeigt schon deutlich, dass es beim MVC-Pattern um eine dreigliedrige Entwicklung handelt. Das Modell enthält die darzustellenden Daten und gegebenenfalls die Geschäftslogik. Die View ist für die Darstellung der Daten aus dem Modell und für die Entgegennahmen der Benutzerinteraktionen zuständig und der Controller verwaltet eine oder mehrere Views und nimmt von diesen die Benutzerinteraktionen entgegen, wertet diese aus und reagiert entsprechend darauf. Alle drei Teile des MVC-Patterns sind unabhängig voneinander implementiert und können ohne auftretende Probleme getauscht werden.

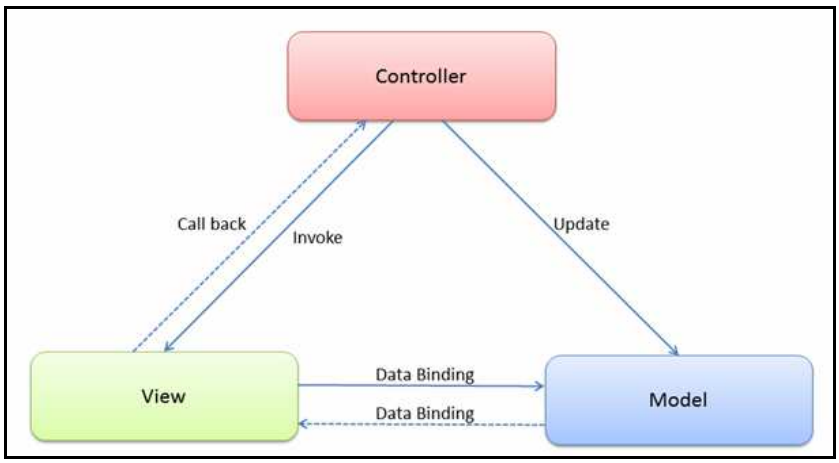


Abbildung 100: MVC Pattern im Überblick (Quelle: http://mnikoo.net/tag/mvc/)

In der ersten fli4l-GUI wird das MVC-Pattern um eine weitere Schicht erweitert. Diese 4. Schicht wird als ViewLogic bezeichnet und beinhaltet die komplette Oberflächenlogik und Oberflächendatenstruktur. Damit wird die Oberflächenlogik[22] aus der View herausgezogen und ist dadurch ebenfalls ein unabhängiger Bestandteil, der geändert oder ausgetauscht werden kann, ohne Probleme in der designten Oberfläche zu verursachen. Der Controller unterscheidet sich in der Erweiterung nur darin, dass er die Benutzerinteraktionen nicht mehr von der View erhält, sondern diese jetzt an die ViewLogic geht. Die ViewLogic übernimmt die Abarbeitung der Oberflächenlogik und delegiert alles andere an den Controller. Der Controller löst die Bearbeitung der Geschäftslogik[23] aus und delegiert diese an die ModelFacade. Diesezentralisiert alle Anfragen an das Modell und delegiert diese dann weiter. Der Einsatz der ModelFacade führt dazu, dass man in der Zukunft mehrere Modells oder Controller haben kann und das der Controller ohne die Klassen, ihre Beziehungen, und Abhängigkeiten vom Modell zu kennen, auf diese zugreifen kann. (siehe auch Kapitel Facade-Pattern)
Die Vorteile der von uns verwendeten 4-Schichten-Architektur liegen in der vollständigen Entkopplung der View von der Oberflächen- und der Geschäftslogik und der klaren Abgrenzung der Fach-

[22] Die Oberflächenlogik für die Bedienung der Oberfläche.
[23] Die Geschäftslogik wird auch Anwendungslogik genannt und beinhaltet die Logik die für das Verarbeiten innerhalb der Anwendung nötig ist.

schicht. Dadurch ist die View einfach zu gestalten, da man nur auf das „Aussehen“ achten muss. Außerdem ist ein großer Vorteil dieser Architektur, dass jede Schicht für sich alleine arbeitet und komplett „dumm“ ist. Des Weiteren ist der Austausch sowohl der Oberfläche selbst, als auch der Swing API durch die 4-Schichten-Architektur sehr einfach. Will man nur die Oberfläche austauschen, tauscht man die Views. Möchte man die Swing API durch eine andere, beispielsweise SWT, ersetzen, tauscht man die View und die zugehörige Logic-Schicht aus.

10.3.2. Struktur unserer Anwendung

Wie schon beschrieben besteht die Struktur für die erste GUI aus 4 Schichten: den Views, den „ViewLogics“, dem MainController und der „ModelFacade“ als Schnittstelle zum eigentlichen Model. Um auch die einzelnen Oberflächen voneinander abzugrenzen wird für jede View eine eigene Klasse angelegt. Für jede View-Klasse wird eine „ViewLogic“-Klasse angelegt. Diese enthält die Oberflächenlogik und verbindet die einzelnen Views zu einer miteinander arbeitenden Gesamtoberfläche. Aus diesem Grund sind die „ExceptionViewLogic“ und die „AddRouterViewLogic“ auf die „MainViewLogic“ gerichtet. Die einzelnen „ViewLogic“ sind auf den MainController gerichtet und leiten Benutzerinteraktionen, die mehr als der Oberflächenlogik entsprechen, an diesen weiter. Der MainController delegiert die Geschäftslogik an die ModelFacade und stößt über diese Schnittstelle die Operationen im Model an (Scanner, Parser und Printer). Eine besondere Unterteilung findet bei der „MainView“ statt. Diese wird zusätzlich in die Klassen „MainViewTableArea“ und „MainViewTreeArea“ und „MainViewMenueArea“ aufgeteilt. Die „MainViewTableArea“ stellt den Anzeigebereich dar, die „MainViewTreeArea“ den Tree-Bereich und die „MainViewMenueArea“ den Bereich der Menüleiste (siehe dazu auch Kapitel „View/ Oberfläche“). Diese Aufteilung dient dem einfacheren Erweitern und dem Erhalt der Übersichtlichkeit, auch bei einer komplexen Oberfläche.
Die View-Klassen beinhalten jeweils eine „config“-Methode. Diese konfiguriert die einzelnen Elemente für die Oberfläche. In den „ViewLogics“ sind die Methoden „initLogic“ und „initModel“ enthalten. Die „initLogic“-Methode ist für die Oberflächenlogik zuständig und erstellt die benötigten Listener für diese Aufgabe. Die „initModel“-Methode erzeugtModel-Objekte für die Oberflächeund benutzt dafür den Controller. Wichtig ist, dass die „initModel“-Methode dabei nicht für irgendwelche Seiteneffekte zuständig ist. Der „MainController“ beinhaltet die beiden Methoden „save“ und „addRouter“. Diese beiden Methoden sind für das Abarbeiten der Geschäftslogik zuständig und werden ausgelöst, wenn der Anwender an der Oberfläche den „Add Router“-Button drückt oder in einem Assignmentfile das Kontextmenü der rechten Maustaste aufruft. Der „MainController“ benötigt zum Arbeiten die „ModelFacade“ und diese Fassadedie schon vorhandenen Scanner, Parser und Printer. Dafür delegiert sie die entsprechenden Aufgaben an die Klasse „ModelFacade“. Diese enthält die zwei Methoden „scanAndParser“ und „print“. Die „scanAndParse“-Methode wird für das Hinzufügen von Routern gebraucht und ruft den Scanner und Parser für das Einlesen und Umwandeln der einzelnen Dateien auf. Diese dadurch bereitgestellten Dateien können dann in der GUI angezeigt werden. Die „print“-Methode ruft den Printer für das Ausgeben des geänderten Assignmentfiles auf. Außerdem gibt es noch die zwei Klassen „ApplicationManager“ und „ApplicationManagerViewLogic“. Der „ApplicationManager“ verwaltet die Applikation. Die Klasse „ApplicationManagerViewLogic“ managt die Oberflächenlogik für die Applikation und ist auf die „MainViewLogic“ gerichtet. Der „ApplicationManager“ ist auf den „ApplicationManagerViewLogic“, „MainController“ und „ModelFacade“ gerichtet. Die „MainApplication“ beinhaltet bisweilen nur die main-Methode, um die Anwendung zu starten.

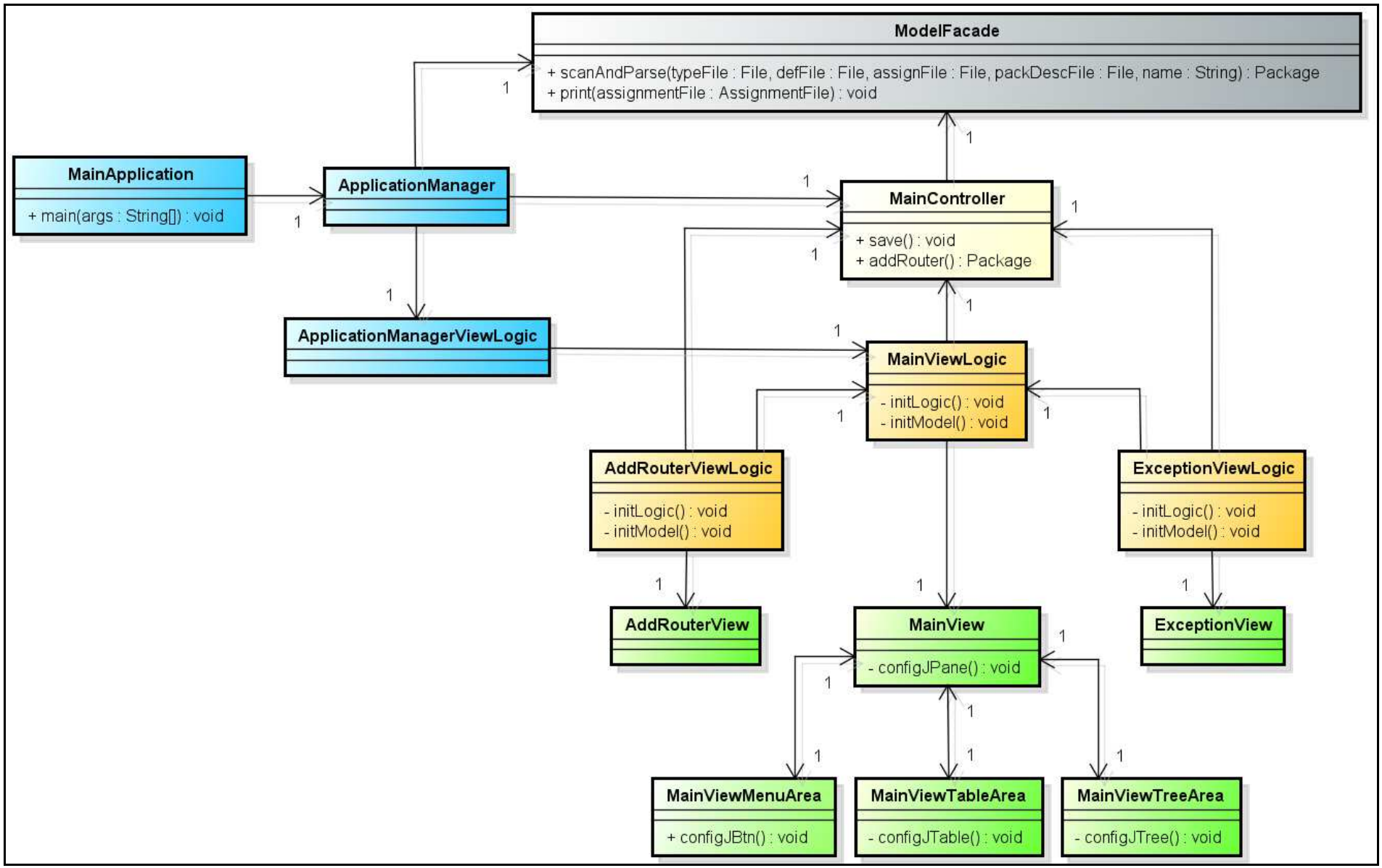


Abbildung 101: UML-Diagramm der ersten GUI

10.3.3. Facade-Pattern

Beim Facade-Pattern definiert man eine vereinfachte Schnittstelle zu einer Menge von Schnittstellen eines Subsystems oder einer Menge von Objekten.

Geht man davon aus, dass man ein Subsystem mit mehreren Klassen und Abhängigkeiten zwischen ihnen hat und mehrere Clients dieses Subsystem oder Teile von ihm nutzen möchten, müsste man sich zunächst mit den verschiedenen Schnittstellen der einzelnen Klassen befassen und die Funktionsweise verstehen. Dieses Vorgehen ist nicht nur sehr zeitaufwendig sondert baut zwangsläufig viele Abhängigkeiten zu verschiedenen Objekten auf und koppelt diese eng an die Klassen des Subsystems.

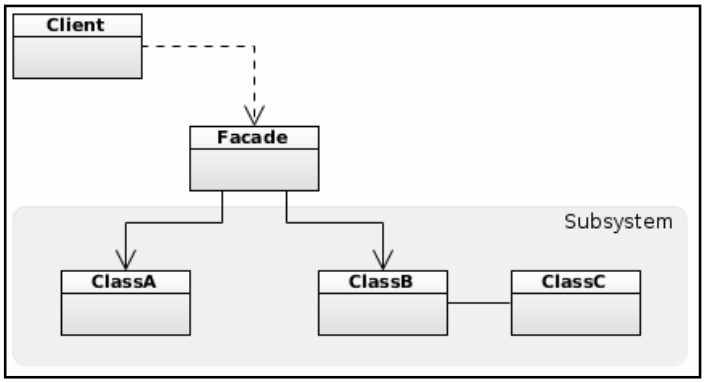


Abbildung 102: Facade-Pattern im Überblick (Quelle: http://javaobsession.wordpress.com/2010/07/23/facade-pattern/)

Das Facade-Pattern wird bei dem Beispiel zwischen die Clients und das Subsystem „geschaltet“. Es sorgt für eine Kapselung des Subsystems und bietet den Clients eine vereinfachte Schnittstelle nach außen an. Das Facade-Pattern delegiert die Clientaufrufe an das Subsystem. Die Clients nutzen das Pattern für den Zugriff auf das System, ohne die Klassen, ihre Beziehungen und Abhängigkeiten kennen zu müssen.

Der Vorteil des Facade-Patterns ist, dass dadurch die lose Kopplung gefördert wird, weil mehrere Schnittstellen zu einer zusammengefasst werden, und die Komplexität der erstellten Anwendung sinkt, da sie das Subsystem versteckt. Außerdem kann das Subsystem durch die lose Kopplung einfacher erweitert werden.

10.3.4. View / Oberfläche

Die entwickelte Oberfläche wird aus drei Haupt-Elementen bestehen. Die „MainView“, die „ExceptionView“ und die„AddRouterView“. Die „MainView“ ist die Hauptoberfläche und auf dieser werden die meisten Benutzerinteraktionen durchgeführt. Die „ExceptionView“ ist ein Pop-up Fenster,welches erscheint wenn eine Exception (Ausnahme) ausgelöst wird und informiert den Anwender über die aufgetretene Ausnahme. Durch Drücken des „Cancel“-Buttons gelangt der Nutzer wieder zurück zur „MainView“. Die „AddRouterView“ ist ein Dialogfenster, welches erscheint, wenn der Nutzer den „AddRouter“-Button betätigt und mit dem der Anwender Routerverzeichnisse hinzufügen kann.
Die Main-View besteht aus zwei Main-Bereichen. Auf der linken Seite wird eine Baumstruktur angezeigt mit dessen Hilfe sich der Anwender durch die aufgerufenen Router-Konfigurationen navigieren kann und auf der rechten Seite befindet sich der Anzeigebereich.

Die Baumstruktur hat drei Ebenen: Router, Paket und Datei. Die erste Baumebene ist die Routerebene. Hier erhält der Nutzer eine Übersicht über alle eingelesenen Router. Klappt der Nutzer die einzelnen Router auf, gelangt er auf die zweite Ebene, die Paketebene. Hier werden alle Pakete angezeigt die zu

dem entsprechenden Router gehören. Auch diese kann der Nutzer erneut ausklappen und gelangt so auf die unterste Ebene, die Dateiebene. Auf dieser Ebene werden alle in dem jeweiligen Paket vorhandenen Dateien angezeigt und können von dem Anwender aufgerufen werden. Wenn eine Datei aufgerufen wird, erscheint der Inhalt im Anzeigebereich auf der rechten Seite der Oberfläche. Die Inhalte der einzelnen Dateien werden in einer Tabellenstruktur angezeigt. Handelt es sich bei dem Inhalt um ein Assignmentfile, so ist die rechte Spalte (value) änderbar.

Sowohl im Anzeigebereich, als auch im Tree-Bereich gibt es optional eine horizontale und eine vertikale Scrollbar. Die Scrollbars erscheinen nur bei Bedarf. Dieser Bedarf tritt auf, wenn die Breite und/ oder Länge der jeweiligen Bereiche nicht für eine vollständige Anzeige ausreichend sind.

Zusätzlich zu den zwei Bereichen haben wir noch einen Bereich für die Menüleiste. In der Menüleiste ist momentan der „AddRouter“-Buttonzu finden. Mit dem „AddRouter“-Button kann der Anwender ein Routerverzeichnis hinzufügen. Um Änderungen an einem Assignmentfile zu speichern, steht dem Nutzer im Kontextmenü ein „Save Assigns“-Button zur Verfügung. Das Kontextmenü erscheint, wenn man bei einem Assignmentfile die rechte Maustaste drückt.

Die „AddRouterView“ ist ein Dialogfenster für das Hinzufügen von Routerverzeichnissen. Das Dialogfenster besteht aus einem Eingabebereich für den Routernamen und einem Sourcepath-Bereich. Im Sourcepath-Bereich kann der Nutzer entweder per Hand den Pfad zu einem Verzeichnis hineinschreiben oder über den „Search Path“- Button die Dateistruktur aufrufen und dort den Pfad zu dem gewünschten Verzeichnis auswählen. Des Weiteren enthält die „AddRouterView“ noch zwei Button: Einen „Okay“-Button zum Bestätigen der Eingaben und einen „Cancel“-Button zum Abbrechen.

10.3.5. Toolkitauswahl

Die fli4l-GUI wird mit dem Toolkit Java Swing implementiert. Swing ist eine Programmierschnittstelle (API) und Bibliothek zum Programmieren von grafischen Oberflächen. Swing wurde von Sun Microsystems für die Programmierspreche Java entwickelt und ist Bestandteil der Java-Runtime. Swing baut auf dem Abstract Window Toolkit (AWT) auf und ist objektorientiert aufgebaut. Des Weiteren eignet es sich gut für die Entwicklung komplexer Anwendungen. Ein weiterer großer Vorteil der Swing API ist, dass sie von Haus aus schon auf das MVC-Pattern setzt. So hat zum Beispiel jede JList ein ListModel. Bei Swing kann allgemein gesagt werden, dass jede XYZ-Komponente ein XYZ-Model hat.

Hauptkonkurrent von Swing ist SWT(Standard Widget Toolkit). Das Abstract Window Toolkit, auf dem Swing aufbaut, ist eine schon in die Jahre gekommene Programmierschnittstelle, die zwar sehr schlank und schnell ist, aber auch wenige Komponenten beinhaltet. Mit Swing wurde die AWT API erweitert und es entstand eine sehr mächtige Bibliothek zum Gestalten von grafischen Oberflächen, die jedoch auch sehr viele Ressourcen benötigen. SWT ist eine von IBM entwickelte API und Bibliothek zum Erstellen von grafischen Oberflächen mit dem Ziel, zur damaligen Zeit, eine Alternative zu AWT anzubieten. SWT wurde lange Zeit als die schnellste GUI API bezeichnet, doch Untersuchungen haben gezeigt, dass SWT sich in der Geschwindigkeit nicht groß von Swing unterscheidet. SWT und Swing haben gemeinsam, dass sie plattformunabhängig sind. Ansonsten ist SWT native, sprich SWT läuft ohne einen Emulator, der es erst umwandeln müsste, und Swing komfortable, da man mit Swing von jeder Komponente erben kann. Die Entscheidung Swing statt SWT zu verwenden lag hauptsächlich an vorhandenen Kenntnissen im Umgang

mit Swing innerhalb der Gruppe. Aber auch das Swing schon eine vorhandene MVC-Struktur mitbringt. Die MVC-Struktur hat SWT von sich aus nicht.

Aufbau einer Swing-Anwendung

Für das Entwickeln von Swing-Anwendungen benötigt man immer einen Top-Level Container. Der Top-Level Container kann entweder ein leeres Fenster sein oder die Applet-Oberfläche, dem man in der weiteren Entwicklung weitere Swing Container oder Components hinzufügen kann. Steuerelemente wie Tasten, Eingabefelder, Auswahlknöpfe usw. können nicht direkt dem Top-Level Container hinzugefügt werden, sondern nur dem Content Pane, welches zuvor erstellt werden muss. Beim Top-Level Container wird für Java-Applets *javax.swing.JApplet* und für Java-Applications *javax.swing.JFrame* verwendet.

Wie die einzelnen Elemente, die man einem Container hinzufügt, angeordnet werden, hängt von den verwendeten Layout Managern ab. Jeder Container verwendet einen Standard Layout Manager, der die Anordnung festlegt. Layout Manager sind unter anderem die Klassen *java.awt.FlowLayout*, *java.awt.BorderLayout* und *java.awt.GridLayout.* Wie man sieht verwendet Swing an dieser Stelle die gleichen Layout Manager wie AWT. Darüber hinaus gibt es aber auch Swing-spezifische Layout Manager wie beispielsweise *javax.swing.SpringLayout*. Das SpringLayout positioniert die Elemente gemäß der angegebenen Constraints (Einschränkungen). Jedes Constraints ist ein Spring-Exemplar, das den horizontalen und vertikalen Abstand zwischen zwei Komponenten definiert.

Eine weitere Möglichkeit ist die Erstellung von Swing-Anwendungen ohne einen Layout Manager zu verwenden. Dabei müssen alle Container und Komponenten pixelgenau ausgerichtet werden. Diese Möglichkeit gibt einem noch mehr Flexibilität, ist aber auch deutlich aufwendiger als die Verwendung eines Layout Managers. Des Weiteren ist diese Möglichkeit nicht mehr zeitgemäß, da sich die Bildschirmgrößen immer wieder ändern und eine Anwendung unabhängig der Bildschirmgröße einheitlich dargestellt werden soll.

10.4. Implementierung

10.4.1. Die Oberfläche / View-Schicht

Wie bereits erwähnt haben sich die ersten Oberflächenentwürfe der Analysephase (siehe Kapitel 10.3) geändert, das ist einerseits der SWING-Technologie, andererseits der Benutzerfreundlichkeit geschuldet. Die implementierten Oberflächen zeigen die nächsten Abbildungen (bereits mit Daten gefüllt).

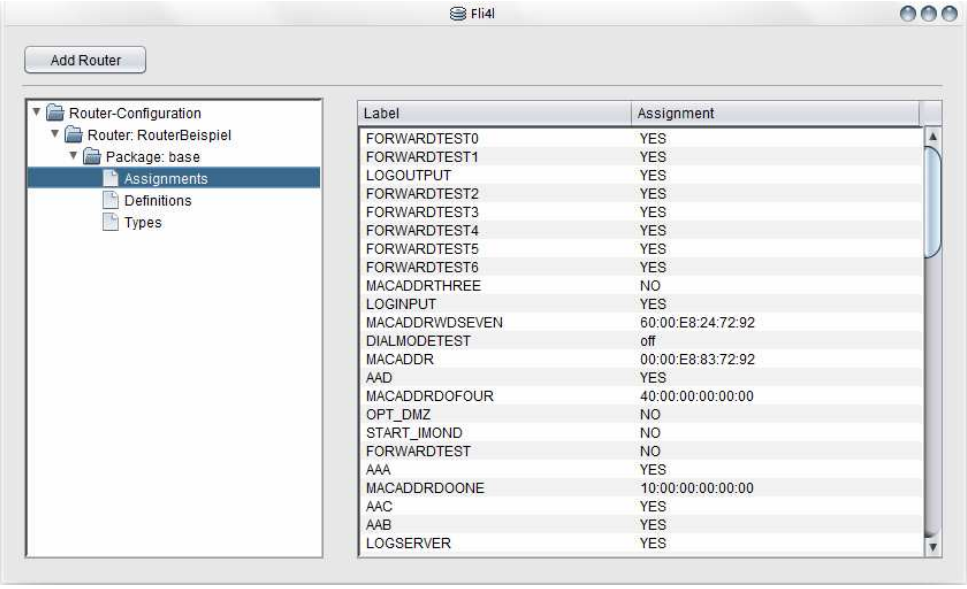


Abbildung 103: Die implementierte Haupt-Oberfläche

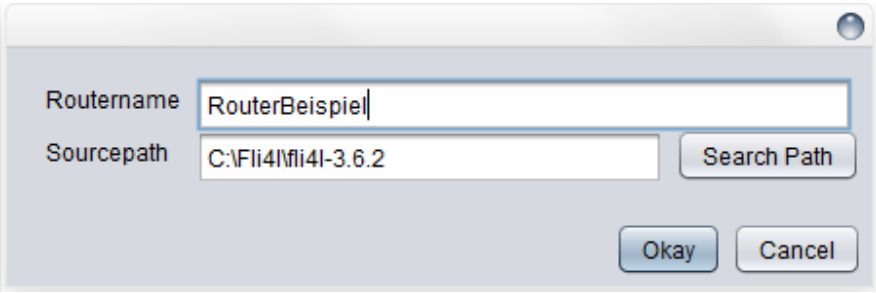


Abbildung 104: Die implementierte addRouter-View

Die Implementierung jeder View-Klasse verfolgt eine gewisse Struktur, die zu Übersichtlichkeit und Verständlichkeit führt. Diese sei am Beispiel der kurzen, aber aussagekräftigen „MainViewTableArea“ gezeigt:

```
public MainViewTableArea
(final MainViewmainView)
{
        super();
        this.mainView = mainView;
        this.table = new JTable();
        this.scrollPaneTable = new JScrollPane();
        this.configJTable();
        this.configJScrollPane();
}
```

Im Konstruktor werden die benötigten Elemente instanziiert und den Feldern zugewiesen.
Danach werden Konfigurations-Methoden für die erstellten Elemente aufgerufen.
Mit Ausnahme der Klasse „MainView“ muss im Konstruktor jeder View Klasse die „MainView“ ü-

bergeben werden, damit diese referenziert werden kann.
Die „MainView“ instanziiert die anderen View-Klassen.

```
private void configJScrollPane() {

this.mainView.getSlRightTablePanel().putConstraint(SpringLayout.NORTH,this.scrollPaneTable,  0,
SpringLayout.NORTH,this.mainView.getRightTablePanel());
this.mainView.getSlRightTablePanel().putConstraint(SpringLayout.EAST,
this.scrollPaneTable, 0, SpringLayout.EAST, this.mainView.getRightTablePanel());
this.mainView.getSlRightTablePanel().putConstraint(SpringLayout.SOUTH,
this.scrollPaneTable, 0, SpringLayout.SOUTH, this.mainView.getRightTablePanel());
this.mainView.getSlRightTablePanel().putConstraint(SpringLayout.WEST,
this.scrollPaneTable, 0, SpringLayout.WEST, this.mainView.getRightTablePanel());

this.mainView.getRightTablePanel().add(this.scrollPaneTable);
}
```

In den Konfigurations-Methoden für die Umgebungen (Panes) werden die Constraints für das Spring-Layout gesetzt. Der erste Constraint besagt dass die nördliche-Seite des Elements scrollPaneTable (eine Umgebung mit Scroll-Bars) genau 0Px von der nördlichen-Seite des Elements rightTablePane sein soll. Es werden noch drei weitere Constraints für die restlichen Himmelsrichtungen angegeben. In diesem Beispiel füllt die Scroll-Bar-Umgebung das rightTablePane voll aus. Am Ende wird die Scroll-Umgebung noch der rightTablePane-Umgebung hinzugefügt (erst dann sind die Cosntraints wirksam). Die Constraints sind immer in der Reihenfolge „North, East, South, West“ des Quellelementes angegeben, um wiederkehrende Strukturen zu schaffen.

```
private void configTable() {
this.table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
this.scrollPaneTable.setViewportView(this.table);
this.table.setBorder(newBevelBorder(BevelBorder.RAISED, Color.DARK_GRAY, Color.GRAY,
Color.LIGHT_GRAY, Color.WHITE));
}
```

In den Konfigurations-Methoden für die angezeigten Elemente (Buttons, Tabellen, Bäume etc.) sind keine Spring-Constraints von Nöten, da die Anordnung der Elemente durch die Panes geschieht auf denen sie liegen. Allerdings ist eine Anordnung per Spring-Constraints möglich, falls gewünscht. In diesem Fall werden nur spezifische Attribute für die Tabelle gesetzt.

An diesem Beispiel sieht man sehr ausdrücklich, dass die Oberfläche vollkommen frei von Logik und Datenelementen ist. Es werden nur sichtbare Elemente erzeugt und angeordnet.

10.4.2. Die Oberflächenlogik / ViewLogic-Schicht

Auch die Klassen der ViewLogic-Schicht verfolgen einen ähnlichen Aufbau, um Komplexität zu minimieren.

```
public MainViewLogic(final MainController controller) {
```

```
        this.controller = controller;
        this.mainView = new MainView();
        this.mainView.getjFrame().setVisible(true);
        initLogic();
        initModelData();
}
```

Um Klassen der Logik-Schicht zu instanziieren ist eine Referenz auf den „MainController“ notwendig, denn an diesen müssen eventuelle Oberflächenaktionen delegiert werden. Die ViewLogic-Klassen sind für das Instanziieren ihrer namensähnlichen View-Klassen zuständig. In diesem Fall wird die „MainView“ instanziiert und das Hauptfenster sichtbar gemacht. Danach werden Methoden für das initialisieren von Logik und Oberflächendaten aufgerufen.

```
private void initModelData() {
this.root = new DefaultMutableTreeNode(basic.GuiConstants.ROUTERCONFIG);
this.modelTree = new DefaultTreeModel(this.root);
this.getMainView().getTreeArea().getTree().setModel(this.modelTree);
}
```

An dieser Stelle werden die nötigen Datenelemente für die Oberflächenklassen erzeugt und den Oberflächenelementen zugewiesen. Ab diesem Moment können jede Daten aufnehmen und anzeigen. In diesem Beispiel wird ein Standard-Modell für Bäume instanziiert und die Wurzel hinzugefügt. Das Model wird im letzten Befehl dem Oberflächenelement, dem JTree zugeordnet.

```
private void initLogic() {
final MouseListener mlBtnAddRouter = new MouseAdapter() {
        @Override
        public void mouseClicked(final MouseEvent arg0) {
                final AddRouterViewLogic addRouterView =
                new AddRouterViewLogic(getController(), mainViewLogicThis);
                addRouterView.getAddRouterView().setVisible(true);
        }
};
this.getMainView().getMainViewMenueArea().getBtnAddRouter()
.addMouseListener(mlBtnAddRouter);
}
```

Hier werden nötige Logik-Schritte implementiert, wie z.B. das Hinzufügen von “ listenern“ und die Reaktion auf diese.In diesem Beispiel wird ein MouseListener erstellt, der auf das Event mouseClicked reagiert. Ist das Event aufgetreten, wird eine AddRouterViewLogic erzeugt und angezeigt. Am Ende wird der „listener“ noch einem Element zugeordnet, auf dessen Events er reagieren soll, in diesem Fall dem Button „AddRouter“. In anderen Fällen wie dem Speichern oder dem Hinzufügen von Routern(hier wird nur das Fenster dazu geöffnet), werden Anfragen an den MainController delegiert.

10.4.3. Der MainController / Controller-Schicht

Der MainController nimmt alle Anfragen der View-Logic-Schicht entgegen, die Auswirkungen auf die Geschäftslogik/Daten haben. Also auf die Daten des fli4l-Models. Im MainController sind die Methoden noch so benannt, wie die Funktionen auf der Oberfläche. Im MainController wird nur an die ModelFacade delegiert. Er ist quasi die äußere Schnittstelle der GUI.

10.4.4. Die ModelFacade / Model-Schicht

Die ModelFacade ist die äußere Schnittstelle des bestehenden fli4l-Models. Diese bietet benötigte Funktionen der bestehenden fli4l-Logik nach außen aufbereitet an. So muss z.B. der Aufrufer nicht wissen, in welcher Reihenfolge Scanner und Parser gestartet werden müssen.

10.4.5. Das Datenmodell der Table und WrapperKlassen

Um die Anforderung der Darstellung der Einträge der fli4l-Konfigurationsdateien umzusetzen wurde ein eigenes Oberflächendatenmodel für das Oberflächenelement JTable implementiert. Auch dazu soll kurz beispielhaft gezeigt werden, wie es umgesetzt wurde. Die weiteren Datenmodelle sind strukturell gleich aufgebaut. Des Weiteren wurden für die bestehenden fli4l-Konfigurations-Dateien Wrapper-Klassen angelegt, die die bestehenden Files kapseln.

```
private final AssignmentFile assignmentFile;
private final String path;

public AssignmentFileWrapper
(final AssignmentFile assignmentFile, final String path) {
this.assignmentFile = assignmentFile;
this.path = path;
}
@Override
public String toString() {
return basic.GuiConstants.ASSIGNMENTSTEXT;
}
```

Die Wrapper-Klassen enthalten eine Referenz auf das jeweilige zu kapselnde File und den Datei-Pfad zu diesem. Dieser Datei-Pfad wird auch beim Speichern an den Printer übergeben. Die toString()-Methode wird überschrieben, damit die Wrapper-Klassen an der Oberfläche verständlich dargestellt werden können. Die Wrapper-Klassen dienen der Darstellung im Baum und ermöglichen zugleich ein kontextsensitives Reagieren.

```
public AssignmentTableDataModel(final List<VariableAssignment> data) {
super(basic.GuiConstants.ASSIGNTABLEHEADER);
this.data = wrapDate(data);
Collections.sort(this.data);
}
@Override
publicbooleanisCellEditable(finalint r, finalint c) {
return c == 1;
```

```
}
@Override
public Object getValueAt(finalint r, finalint c) {
        finalVariableAssignment assignment = this.data.get(r).getAssignmentFile();
        String retString = "";
        switch (c) {
        case 0:
                retString = assignment.getNameOfAssignment();
                break;
        case 1:
                retString = assignment.getValue();
                break;
        default:
                break;
        }
Return retString;
}
```

Im Konstruktor wird ein Array mit den Spaltenüberschriften an den super-Konstruktor übergeben und es werden die übergebenen Daten in das Feld gesetzt (nach dem „Umhüllen“ mit der WrapperKlasse und dem Sortieren). Die überschriebene Methode isCellEditable definiert, wann eine Zelle editierbar ist, in diesem Fall ist nur Colomn 1 (die erste ist Colomn 0) editierbar. Also die Value-Spalte. GetValueAt legt fest, welche Daten des Files wie angezeigt werden sollen. In Colomn 0 wird der Name angezeigt, in Colomn 1 der Value.

Solche Tabellenmodelle werden bei der Instanziierung durch Übergabe der Daten bereits gefüllt.

10.4.6. Testfälle

Neben den reinen Oberflächentests, die anhand der Funktionen manuell getestet werden müssen, sind im Testordner unter dem Paket „gui“ junit-Testfälle geschrieben. Diese decken die implementierte Logik ab. Dazu gehören die Datenmodelle, die Delegation von Funktionen und das Verhalten der Wrapper-Klassen. Sämtliche ViewLogic-Testklassen sind auskommentiert, da sie derzeit zu Problemen im Jenkins führen. Lokal laufen diese problemlos.

10.5. Funktionen

Funktion 1 : Programm starten

Szenario:

1. Der User startet das Programm.

Häufigkeit: selten

Funktion 2 : Router hinzufügen

Szenario:

1. Der Nutzer gelangt durch das Klicken auf die Schaltfläche „AddRouter“ zu einer zusätzlichen Dialogansicht, mit welcher er einen neuen Router hinzufügen kann.

Häufigkeit: normal

Funktion 3 : Pfadauswahl für Hinzufügen des Routers

Szenario:

1. Der Nutzer kann den Pfad zum Router entweder direkt angeben oder aus dem Dateisystem heraussuchen.

Häufigkeit: normal

Funktion 4 : Namensvergabe für Router

Szenario:

1. Der Nutzer muss dem Router einen Namen geben, welcher nach dem Hinzufügen des Routers in der Routerliste sichtbar ist.

Häufigkeit: normal

Funktion 5 : Bestätigen des Hinzufügen des Routers

Szenario:

1. Um den Router letztendlich hinzuzufügen, muss in der Dialogansicht das Hinzufügen bestätigt werden.

Häufigkeit: normal

Mögliches Fehlverhalten:

1. Das Feld zur Pfadangabe und/oder das Feld des Namens ist leer.
2. Der angegebene Pfad ist inkorrekt.

Funktion 6 : Abbrechen des Hinzufügen des Routers

Szenario:

1. Wenn der Nutzer den Router nun doch nicht mehr hinzufügen möchte, kann er den Vorgang abbrechen. Hierbei wird die Dialogansicht wieder geschlossen und die angegebenen Informationen verworfen.

Häufigkeit: selten

Funktion 7 : Erweiterung der Baumstruktur der Routerliste

Szenario:

1. Per Doppelklick auf den jeweiligen Router werden die zugehörigen Pakete sichtbar.
2. Ein weiterer Doppelklick auf das jeweilige Paket zeigt dem Nutzer die zum Paket gehörenden Dateien an.

Häufigkeit: sehr häufig

Funktion 8 : Öffnen der Detailansicht zu einer Datei

Szenario:

1. Per Doppelklick auf eine Datei in der Routerliste, erhält der Nutzer eine detaillierte und strukturierte Ansicht der Informationen aus der Datei.

Häufigkeit: sehr häufig

Funktion 9 : Editierung der Zuweisungen

Szenario:

1. Dem Nutzer muss es ermöglicht werden, bereits vorhandene Zuweisungen zu editieren.

Häufigkeit: häufig

Funktion 10 : Speichern der Änderungen

Szenario:

1. Der Nutzer kann bei Bedarf die derzeitig ausgewählte Datei speichern.
2. Dies sollte über das Kontextmenü der zu speichernden Datei ermöglicht werden.

Häufigkeit: normal

Mögliches Fehlverhalten:

1. Der Pfad ist nicht mehr vorhanden
2. Schreibrechte fehlen

Funktion 11 : Horizontales und vertikales Scrollen

Szenario:

1. Sollten die Informationen mehr Platz einnehmen, als zur Verfügung steht, so soll der Nutzer mit Hilfe von Scrollbalken die erst nicht sichtbaren Informationen einsehen können.

Häufigkeit: sehr häufig

Funktion 12 : Programm beenden

Szenario:

1. Bei Bedarf muss das Programm auch beendet werden können.

Häufigkeit: selten

10.6. Ausblick

Wie in der Analyse erwähnt ist in Zukunft ein persistentes Abspeichern des Zustandes der Anwendung denkbar. Dazu muss nur noch in der Klasse MainApplication gesteuert werden, ob bereits eine Daten-

struktur vorliegt (z.B. in Form von Objectdateien im Ordner der Anwendung) oder ob es der erste Start der Anwendung ist. Entweder wird dann ein neuer ApplicationManager instanziiert oder der persistente eingelesen und gesetzt. Außerdem müsste beim Schließen der Anwendung der bisherige Status in Form der erwähnten Objektdatei abgelegt werden.

Dieses ist aber nur eine Idee und bedarf noch stärkerer Analyse.

Abbildungsverzeichnis

Abbildung 1: Testumgebung für das Integrationsprojekt 9
Abbildung 2: Programm zum Erstellen von fli4l-Builds 19
Abbildung 3: Programm während der Erstellung eines fli4l-Builds 20
Abbildung 4: Einhängen eines neuen virtuellen Diskettenlaufwerkes 24
Abbildung 5: Einstellung für ein Disketten-Image 25
Abbildung 6: Eclipse 27
Abbildung 7: FindBugs 27
Abbildung 8: PMD 28
Abbildung 9: Checkstyle 29
Abbildung 10: Ant 30
Abbildung 11: Subversion 30
Abbildung 12: Jenkins 31
Abbildung 13: Verwendung Findbugs 32
Abbildung 14: Verwendung PMD 33
Abbildung 16: Checkstyleanzeige 34
Abbildung 17: Verwendung Cobertura 34
Abbildung 18: Repository View aufrufen 35
Abbildung 19: Repository Location hinzufügen 35
Abbildung 20: Repository Location 35
Abbildung 21: Checkout Projekt 36
Abbildung 22: Synchronisieren der Daten 36
Abbildung 23: Änderungen im Repository 37
Abbildung 24: Source-Code-Management 38
Abbildung 25: Buildverfahren 38
Abbildung 26: Übersicht Jenkins 39
Abbildung 27: Projektansicht 40
Abbildung 28: Änderungen am Repository 40
Abbildung 29: CheckStyle Ergebnis 41
Abbildung 30: Testabdeckung 41
Abbildung 31: Buildstatus 42
Abbildung 32: Gesamtes Klassenmodel 45
Abbildung 33: Scanner 46
Abbildung 34: Parser 46
Abbildung 35: Printer 47
Abbildung 36: Pakete 48
Abbildung 37: GUI 49
Abbildung 38: Ordnerstruktur fli4l 50
Abbildung 39: Auszug aus check/base.txt (Variablendefinition) 51
Abbildung 40: Auszug aus check/base.reg (Variablentyp) 52
Abbildung 41: Auszug aus config/base.txt (Variablenzuweisung) 52

Abbildung 42: UML2 Klassendiagramm (fachlich) 53
Abbildung 43: UML2-Klassendiagramm (technischer Entwurf) 56
Abbildung 44: Projektstruktur 57
Abbildung 45: Zusammenhang Scanner, Parser und Printer 58
Abbildung 46 - Anwendungsfalldiagramm 59
Abbildung 47 - Funktionsweise des Scanners 60
Abbildung 48 - Klassendiagramm Scanner 61
Abbildung 49 - Klassendiagramm Zustände 63
Abbildung 50 - Klassendiagramm Symbole 65
Abbildung 51 - Zustände Dateitypscanner 66
Abbildung 52: Zustände Definitionsdateiscanner 68
Abbildung 53 - Zustände Zuweisungsdateiscanner 69
Abbildung 54: Alte Version des Klassendiagramms des Modells aus der Gruppe Konfiguration 1 71
Abbildung 55: Datenfluss von der Konfigurationsdatei bis zum Parser 71
Abbildung 56: Zustandsdiagramm des AssignmentParsers 72
Abbildung 57: Zustandsdiagramm des TypeParsers 72
Abbildung 58: Zustandsdiagramm des DefinitionParsers 73
Abbildung 59: Ergänzte Version des Klassendiagramms des Konfigurationsmodells 74
Abbildung 60: Klassendiagramm aller Parser-Klassen 75
Abbildung 61: Klassen GlobalFunctions und PackageParser 76
Abbildung 62: Klassendiagramm des TypeFileParsers 77
Abbildung 63:Klassendiagramm des DefinitionFileParsers 78
Abbildung 64: Klassendiagramm des AssignmentFileParsers 79
Abbildung 65: Variablen Definition in fli4l 82
Abbildung 66: Variablen Zuweisung in fli4l 82
Abbildung 67: Variablen Typen in fli4l 82
Abbildung 68: Änderungen im Klassendiagramm 84
Abbildung 69: Modell – Fachklassendiagramm 89
Abbildung 70: Modell - Klassendiagramm 90
Abbildung 71: Scanner - Fachklassendiagramm 91
Abbildung 72: Scanner Zustände - Fachklassendiagramm 91
Abbildung 73: Blockstruktur - Fachklassendiagramm 92
Abbildung 74: Symbole - Fachklassendiagramm 93
Abbildung 77: Zustandsdiagramm Scanner 94
Abbildung 78: Ablaufdiagramm Scanner 95
Abbildung 79: Scannerablauf PackageDescriptionInitalState 96
Abbildung 80: Scannerablauf WhitespaceState 96
Abbildung 81: Scannerablauf CommentState 97
Abbildung 82: Scannerablauf ContentState 97
Abbildung 83: Scanner Klassendiagramm 98
Abbildung 84: Symbol - Klassendiagramm 99
Abbildung 85: Parser - Fachklassendiagramm 101
Abbildung 86: Parser Zustände - Fachklassendiagramm 102

Abbildung 87: Parser - Ablaufdiagramm 103
Abbildung 88: Parser – Zustandsdiagramm 104
Abbildung 89: Initialsierungszustand – Ablaufdiagramm 104
Abbildung 90: Variablenzustand - Ablaufdiagramm 105
Abbildung 91: Wertzustand: Ablaufdiagramm 106
Abbildung 92: Pfadzustand – Ablaufdiagramm 107
Abbildung 93: Optionszustand – Ablaufdiagramm 108
Abbildung 94: Kommentarzustand - Ablaufdiagramm 109
Abbildung 95: Formatversionszustand – Ablaufdiagramm 110
Abbildung 96: Parser - Klassendiagramm 111
Abbildung 97: Parserzustände – Klassendiagramm 112
Abbildung 98: Entwurf der MainView 116
Abbildung 99: Entwurf der AddRouterView 116
Abbildung 100: MVC Pattern im Überblick (Quelle: http://mnikoo.net/tag/mvc/) 117
Abbildung 101: UML-Diagramm der ersten GUI 119
Abbildung 103: Die implementierte Haupt-Oberfläche 123
Abbildung 104: Die implementierte addRouter-View 123

Tabellenverzeichnis

Tabelle 1: IP-Adressbereiche der Netzwerke 10
Tabelle 2: Router der Testumgebung 10
Tabelle 3: Relevante Einstellungen der Konfigurationsdatei "base.txt" vom Router Firewall 12
Tabelle 4: Relevante Einstellungen der Konfigurationsdatei "dns_dhcp.txt" vom Router Firewall 13
Tabelle 5: Relevante Einstellungen der Konfigurationsdatei "dhcp_client.txt" vom Router Firewall .. 13
Tabelle 6: Relevante Einstellungen der Konfigurationsdatei "base.txt" vom Router R0 15
Tabelle 7: Relevante Einstellungen der Konfigurationsdatei "dns_dhcp.txt" vom Router R0 16
Tabelle 8: Relevante Einstellungen der Konfigurationsdatei "base.txt" vom Router R1 17
Tabelle 9: Relevante Einstellungen der Konfigurationsdatei "dns_dhcp.txt" vom Router R1 18
Tabelle 10: Legende für die Testfälle 21
Tabelle 11: ICMP-Testfälle 21
Tabelle 12: TCP-Testfälle für http 22
Tabelle 13: TCP-Testfälle für ftp 22
Tabelle 14: DNS-Testfälle für Ping 22
Tabelle 15: Standard-Zeichenklassen 55
Tabelle 16: Konfigurationsdateien von fli4l 70
Tabelle 17: Zeilenaufbau der ersten Spalte 86
Tabelle 18: Zeilenaufbau der folgenden Spalten 86
Tabelle 19: Übersicht der Optionen 88

Glossar

Active Directory:
Name des Verzeichnisdienstes von Microsoft Windows Server. Ermöglicht eine Gliederung des Netzwerks in Form von Benutzer, Benutzergruppen, Computer, Server, uvm.

Applikationsebene (ISO / OSI):
Eine Schicht des genormten ISO/OSI-Modells, die diverse Netzwerkprotokolle beschreibt. Über der Applikationsebene können Programme mit einem Netzwerk kommunizieren. Bekannte Protokolle sind hier z.B. HTTP oder FTP.

Bluetooth:
Entwickelter Standard zur Funkübertragung. Wird u.a. in mobilen Geräten eingesetzt, wie z.B. Handys oder Notebooks.

Client:
Ein Computerprogramm das Kontakt zu einem Server aufnimmt und dessen Dienste nutzt. Der Server übernimmt hierbei die Aufgabe bestimmte Dinge für den Client zu lösen.

Dienste:
Beschreibt eine technische Einheit, die zusammenhängende Funktionen bündelt und bereitstellt. Typischer Dienst bei einem Windowssystem wäre z.B. die Druckerwarteschlange.

Firewall:
Software, die den Datenverkehr eines Netzwerks überwacht und ggf. bei unbefugtem Netzwerkzugriff von intern oder extern diesen blockiert.

Host:
Ein im Netzwerk eingebundener Computer mit eigener IP-Adresse, der Clients bedient oder Server beherbergt. Kann sich auch um einen Router, Switch oder anderes Netzwerkgerät handeln, das als Knotenpunkt des Informationsaustauschs in einem Netzwerk dient.

Intranet:
Lokales Rechnernetz mit, im Gegensatz zum Internet, stark eingeschränktem Benutzerkreis.

Internet Control Message Protocol (ICMP):
Netzwerkprotokoll, das dem Austausch von Informationen und Fehlermeldungen dient.

LAN:
Local Area Network, ist ein Rechnernetz, welches sich auf die Umgebung eines Grundstückes beschränkt.

Log-Datei:
Eine Protokolldatei auf einem Rechner.

MAC-Adresse:

Hardwareadresse eines Geräts, zur eindeutigen Identifikation des Geräts in einem Netzwerk. Internet Protocol (IP) Netzwerkprotokoll, das der Netzwerkverbindung von Rechnern dient.

Open Source:
Eine Lizenz für Software, dessen Quelltext öffentlich zugänglich ist und dadurch Weiterentwicklungen fördert. Findet man zumeist bei nicht-kommerzieller Software.

Parser
Ein Parser ist ein Computerprogramm, das für die Zerlegung und Umwandlung einer beliebigen Eingabe in ein für die Weiterverarbeitung brauchbares Format zuständig ist.

Printer
Der Printer ist dafür zuständig, dass aus Objekten Ausgaben erzeugt werden.

Puffer (Buffer)
Puffer (engl. Buffer) sind Speicher für die Zwischenlagerung von Daten. Je nach Aufgabe kann der Puffer unterschiedlich strukturiert sein.

Ein Puffer speichert die Daten in der Regel nur zeitweise.

Scanner
Ein Scanner (Tokenizer) ist ein Computerprogramm zur Zerlegung von Quellcode in Folgen von logisch zusammengehörigen Einheiten, so genannte Token (engl. tokens).

Server:
Ein Server ist ein besonders zuverlässiger Rechner, der anderen Computern (Clients) im Netzwerk Dienste bereitstellt.

Switch:
Ist ein Gerät, welches Netzwerksegmente physisch miteinander verbindet.

Terminal:
Eingabebereich zum Steuern einer Software. Bei Windows-Systemen ist der Begriff „Eingabeaufforderung“ oder „Konsole“ geläufiger.

Virtuelle Maschine:
Virtueller Computer, der nicht aus Hardware sondern nur aus Software besteht. Eine oder mehrere virtuelle Maschinen können gleichzeitig auf einem (Hardware-)Computer laufen. Es läuft parallel zum eigentlichen Betriebssystem auf dem Computer.

Webserver:
Ein Webserver ist ein Dienst, der Webseiten im Netzwerk bereitstellt.

11. Java-Dokumentation

Klassenhierarchie

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

parser.states.assignmentstates.AbstractAssignmentState 148
- parser.states.assignmentstates.AssignmentCommentState 192
- parser.states.assignmentstates.AssignmentIdentifierState 208
- parser.states.assignmentstates.AssignmentInitialState 210
- parser.states.assignmentstates.AssignmentValueState 215

parser.states.definitionstates.AbstractDefinitionState 150
- parser.states.definitionstates.DefinitionCommentState 236
- parser.states.definitionstates.DefinitionDefaultState 238
- parser.states.definitionstates.DefinitionIdentifierState 256
- parser.states.definitionstates.DefinitionInitialState 258
- parser.states.definitionstates.DefinitionOptState 260
- parser.states.definitionstates.DefinitionTypeState 265
- parser.states.definitionstates.DefinitionVariableNState 267

parser.AbstractFileParser 152
- parser.AssignmentFileParser 196
- parser.DefinitionFileParser 242
- parser.PackageDescriptionFileParser 358
- parser.TypeFileParser 521

parser.AbstractPackageParserState 155
- parser.CommentState 224
- parser.FormatVersionState 290
- parser.OptionState 343
- parser.PackageDescriptionInitialState 373
- parser.PathState 408
- parser.ValueState 546
- parser.VariableState 572

model.type.AbstractRefState 157
- model.type.RefResState 438
- model.type.RefUnResState 443

scanner.AbstractState 160
- scanner.AssignmentFileRestState 199
- scanner.CommentState 226
- scanner.ContentState 233
- scanner.DefinitionFileRestState 246
- scanner.DefinitionFileSelectionState 252
- scanner.IdentifierState 308
- scanner.PackageDescriptionInitialState 371
- scanner.RegExpBeginSelectionState1 452
- scanner.RegExpBeginSelectionState2 454
- scanner.RegExpState 456

scanner.RowEndState 468
scanner.SelectionState 486
scanner.StringState 490
scanner.TypeFileRestState 524
scanner.WhitespaceState 581

symbols.AbstractSymbol 163
symbols.ColonSignSymbol 219
symbols.CommentSymbol 228
symbols.EmptySymbol 271
symbols.EndSymbol 274
symbols.EqualSignSymbol 278
symbols.IdentifierSymbol 310
symbols.OptionSymbol 348
symbols.PathSymbol 413
symbols.PlusSymbol 418
symbols.RegExpSymbol 458
symbols.RowEndSymbol 470
symbols.StringSymbol 492
symbols.UnknownSymbol 539
symbols.ValueSymbol 548
symbols.VariableSymbol 574

parser.states.typestates.AbstractTypeState 169
parser.states.typestates.TypeCommentState 510
parser.states.typestates.TypeErrormessageState 514
parser.states.typestates.TypeExpressionState 516
parser.states.typestates.TypeIdentifierState 532
parser.states.typestates.TypeInitialState 534

model.definition.AbstractVariableDefinition 171
model.definition.VariableDefinitionArray 561
model.definition.VariableDefinitionSingle 565

model.type.AbstractVariableType 176
model.type.AnonVariableType 185
model.type.NamedVariableType 333

gui.AddRouterViewLogic 183
main.ApplicationMain 188
main.ApplicationManager 189
main.ApplicationManagerViewLogic 190
model.assignment.AssignmentFile 194
basic.Buffer< T > 217
model.Comment 222
Comparable
gui.wrapperobjects.VariableAssignmentWrapper 557
gui.wrapperobjects.VariableDefinitionWrapper 569
gui.wrapperobjects.VariableTypeWrapper 579
model.Position 421

model.definition.DefaultValue 235

model.definition.HasDefault 300
model.definition.HasNotDefault 302

model.definition.DefinitionFile 240
scanner.DescriptionBlock 269
scanner.OptionBlock 341
scanner.PathBlock 406
scanner.ValueBlock 542
scanner.VariableBlock 559

Exception
gui.GuiException 298
model.type.ReferenceIsUnresolvedException 436
model.type.TypeDoesNotExistException 512
model.type.ValueNotInRangeOfTypeException 544
parser.ParserException 402
printer.PrinterException 428
reader.ReaderException 434
scanner.ScannerException 484

gui.ExceptionViewLogic 283
reader.FileReader 285
gui.wrapperobjects.FileWrapper 286
gui.wrapperobjects.AssignmentFileWrapper 205
gui.wrapperobjects.DefinitionFileWrapper 254
gui.wrapperobjects.TypeFileWrapper 530

gui.wrapperobjects.FileWrapperVisitor 288
parser.states.GlobalFunctions 292
basic.GuiConstants 295
gui.MainController 316
gui.view.MainView 319
gui.MainViewLogic 322
gui.view.MainViewMenueArea 324
gui.view.MainViewTableArea 325
gui.view.MainViewTreeArea 326
basic.ModelConstants 328
model.ModelFacade 330
model.definition.Opt 337
model.packages.Option 339
model.Package 353
model.packages.PackageDescriptionFile 356
symbols.PackageDescriptionFileSymbol 366
symbols.CommentSymbol 228
symbols.EndSymbol 274
symbols.OptionSymbol 348
symbols.PathSymbol 413
symbols.RowEndSymbol 470
symbols.ValueSymbol 548
symbols.VariableSymbol 574

symbols.PackageDescriptionSymbolVisitor 375

parser.InitialsStateSymbolVisitor 313

symbols.PackageDescriptionSymbolVisitorWithTempRule 380
parser.OptionStateSymbolVisitor 345
parser.PathStateSymbolVisitor 410

reader.PackageFileQuartet 386
parser.PackageManager 388
parser.PackageParser 390
reader.PackageReader 393
model.packages.PackageRule 395
gui.wrapperobjects.PackageWrapper 397
parser.ParserConstants 399
model.packages.Path 404
basic.PrinterConstants 424
printer.PrinterUtility 430
basic.ReaderConstants 432
model.type.RefStateVisitor 441
model.type.RegEx 446
model.type.RegExPart 448
model.type.RegExRefPart 461
model.type.RegExStringPart 465

model.type.RegExPartVisitor 450
RuntimeException
symbols.WrongTypeError 583

scanner.Scanner 474
scanner.AssignmentFileScanner 201
scanner.DefinitionFileScanner 248
scanner.PackageDescriptionFileScanner 362
scanner.TypeFileScanner 526

scanner.ScannerConstants 479
basic.Semaphor 488
parser.TempPackageRule 495
parser.tempparserobjects.TempVariableAssignment 498
parser.tempparserobjects.TempVariableDefinition 501
parser.tempparserobjects.TempVariableObject 505
parser.tempparserobjects.TempVariableType 506
basic.TestConstants 509
model.type.TypeFile 518
model.assignment.VariableAssignment 553
model.definition.VariableOptReference 571
model.definition.HasnotOpt 304
model.definition.HasOpt 306

AbstractTableModel
gui.datamodels.AbstractTableDataModel 167
gui.datamodels.AssignmentTableDataModel 212
gui.datamodels.DefinitionTableDataModel 262

gui.datamodels.TypeTableDataModel 536

JDialog
gui.view.AddRouterView 179
gui.view.ExceptionView 281

Klassen-Verzeichnis

Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

parser.states.assignmentstates.AbstractAssignmentState 148
parser.states.definitionstates.AbstractDefinitionState 150
parser.AbstractFileParser 152
parser.AbstractPackageParserState 155
model.type.AbstractRefState 157
scanner.AbstractState 160
symbols.AbstractSymbol 163
gui.datamodels.AbstractTableDataModel 167
parser.states.typestates.AbstractTypeState 169
model.definition.AbstractVariableDefinition 171
model.type.AbstractVariableType 176
gui.view.AddRouterView 179
gui.AddRouterViewLogic 183
model.type.AnonVariableType 185
main.ApplicationMain 188
main.ApplicationManager 189
main.ApplicationManagerViewLogic 190
parser.states.assignmentstates.AssignmentCommentState 192
model.assignment.AssignmentFile 194
parser.AssignmentFileParser 196
scanner.AssignmentFileRestState 199
scanner.AssignmentFileScanner 201
gui.wrapperobjects.AssignmentFileWrapper 205
parser.states.assignmentstates.AssignmentIdentifierState 208
parser.states.assignmentstates.AssignmentInitialState 210
gui.datamodels.AssignmentTableDataModel 212
parser.states.assignmentstates.AssignmentValueState 215
basic.Buffer< T > 217
symbols.ColonSignSymbol 219
model.Comment 222
parser.CommentState 224
scanner.CommentState 226
symbols.CommentSymbol 228
scanner.ContentState 233
model.definition.DefaultValue 235
parser.states.definitionstates.DefinitionCommentState 236
parser.states.definitionstates.DefinitionDefaultState 238
model.definition.DefinitionFile 240
parser.DefinitionFileParser 242
scanner.DefinitionFileRestState 246
scanner.DefinitionFileScanner 248
scanner.DefinitionFileSelectionState 252
gui.wrapperobjects.DefinitionFileWrapper 254

parser.states.definitionstates.DefinitionIdentifierState 256
parser.states.definitionstates.DefinitionInitialState 258
parser.states.definitionstates.DefinitionOptState 260
gui.datamodels.DefinitionTableDataModel 262
parser.states.definitionstates.DefinitionTypeState 265
parser.states.definitionstates.DefinitionVariableNState 267
scanner.DescriptionBlock 269
symbols.EmptySymbol 271
symbols.EndSymbol 274
symbols.EqualSignSymbol 278
gui.view.ExceptionView 281
gui.ExceptionViewLogic 283
reader.FileReader 285
gui.wrapperobjects.FileWrapper 286
gui.wrapperobjects.FileWrapperVisitor 288
parser.FormatVersionState 290
parser.states.GlobalFunctions 292
basic.GuiConstants 295
gui.GuiException 298
model.definition.HasDefault 300
model.definition.HasNotDefault 302
model.definition.HasnotOpt 304
model.definition.HasOpt 306
scanner.IdentifierState 308
symbols.IdentifierSymbol 310
parser.InitialsStateSymbolVisitor 313
gui.MainController 316
gui.view.MainView 319
gui.MainViewLogic 322
gui.view.MainViewMenueArea 324
gui.view.MainViewTableArea 325
gui.view.MainViewTreeArea 326
basic.ModelConstants 328
model.ModelFacade 330
model.type.NamedVariableType 333
model.definition.Opt 337
model.packages.Option 339
scanner.OptionBlock 341
parser.OptionState 343
parser.OptionStateSymbolVisitor 345
symbols.OptionSymbol 348
model.Package 353
model.packages.PackageDescriptionFile 356
parser.PackageDescriptionFileParser 358
scanner.PackageDescriptionFileScanner 362
symbols.PackageDescriptionFileSymbol 366
scanner.PackageDescriptionInitialState 371
parser.PackageDescriptionInitialState 373
symbols.PackageDescriptionSymbolVisitor 375
symbols.PackageDescriptionSymbolVisitorWithTempRule 380
reader.PackageFileQuartet 386

parser.PackageManager 388
parser.PackageParser 390
reader.PackageReader 393
model.packages.PackageRule 395
gui.wrapperobjects.PackageWrapper 397
parser.ParserConstants 399
parser.ParserException 402
model.packages.Path 404
scanner.PathBlock 406
parser.PathState 408
parser.PathStateSymbolVisitor 410
symbols.PathSymbol 413
symbols.PlusSymbol 418
model.Position 421
basic.PrinterConstants 424
printer.PrinterException 428
printer.PrinterUtility 430
basic.ReaderConstants 432
reader.ReaderException 434
model.type.ReferenceIsUnresolvedException 436
model.type.RefResState 438
model.type.RefStateVisitor 441
model.type.RefUnResState 443
model.type.RegEx 446
model.type.RegExPart 448
model.type.RegExPartVisitor 450
scanner.RegExpBeginSelectionState1 452
scanner.RegExpBeginSelectionState2 454
scanner.RegExpState 456
symbols.RegExpSymbol 458
model.type.RegExRefPart 461
model.type.RegExStringPart 465
scanner.RowEndState 468
symbols.RowEndSymbol 470
scanner.Scanner 474
scanner.ScannerConstants 479
scanner.ScannerException 484
scanner.SelectionState 486
basic.Semaphor 488
scanner.StringState 490
symbols.StringSymbol 492
parser.TempPackageRule 495
parser.tempparserobjects.TempVariableAssignment 498
parser.tempparserobjects.TempVariableDefinition 501
parser.tempparserobjects.TempVariableObject 505
parser.tempparserobjects.TempVariableType 506
basic.TestConstants 509
parser.states.typestates.TypeCommentState 510
model.type.TypeDoesNotExistException 512
parser.states.typestates.TypeErrormessageState 514
parser.states.typestates.TypeExpressionState 516

model.type.TypeFile 518
parser.TypeFileParser 521
scanner.TypeFileRestState 524
scanner.TypeFileScanner 526
gui.wrapperobjects.TypeFileWrapper 530
parser.states.typestates.TypeIdentifierState 532
parser.states.typestates.TypeInitialState 534
gui.datamodels.TypeTableDataModel 536
symbols.UnknownSymbol 539
scanner.ValueBlock 542
model.type.ValueNotInRangeOfTypeException 544
parser.ValueState 546
symbols.ValueSymbol 548
model.assignment.VariableAssignment 553
gui.wrapperobjects.VariableAssignmentWrapper 557
scanner.VariableBlock 559
model.definition.VariableDefinitionArray 561
model.definition.VariableDefinitionSingle 565
gui.wrapperobjects.VariableDefinitionWrapper 569
model.definition.VariableOptReference 571
parser.VariableState 572
symbols.VariableSymbol 574
gui.wrapperobjects.VariableTypeWrapper 579
scanner.WhitespaceState 581
symbols.WrongTypeError 583

Dokumentation der Namensbereiche

Paket file

file

Ausführliche Beschreibung

constructor initialize the position with a row and column.

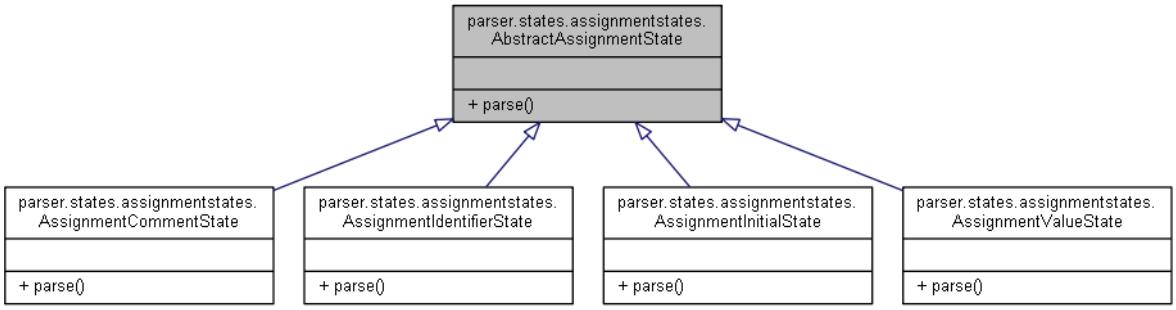
Parameter:

row	is the row of the symbol
column	is the row of the symbol
path	of the datastream expected

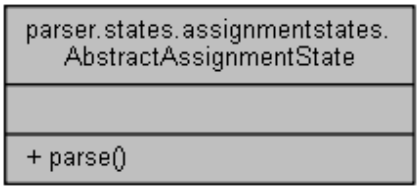
Klassen-Dokumentation

parser.states.assignmentstates.AbstractAssignmentState Klassenreferenz

parser.states.assignmentstates.AbstractAssignmentStateKlassendiagramm für parser.states.assignmentstates.AbstractAssignmentState:



Zusammengehörigkeiten von parser.states.assignmentstates.AbstractAssignmentState:



Öffentliche Methoden

- abstract void parse (TempVariableAssignment tVA, Buffer< AbstractSymbol > buffer, AssignmentFileParser aFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Sandra

Dokumentation der Elementfunktionen

abstract void parser.states.assignmentstates.AbstractAssignmentState.parse (TempVariableAssignment *tVA*, Buffer< AbstractSymbol > *buffer*,

AssignmentFileParser *aFP*) throws InterruptedException, ParserException[pure virtual]

Parameter:

tVA	TempVariableAssignment
buffer	Buffer<Symbol>
aFP	AssignmentFileParser

Ausnahmebehandlung:

InterruptedException	InterruptedException

Rückgabe:

Ausnahmebehandlung:

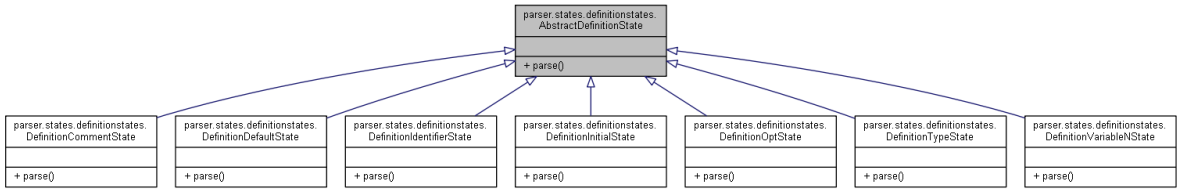
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

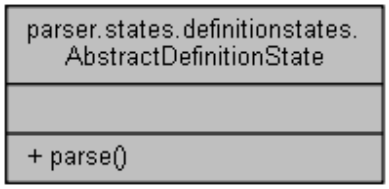
- workspace/ipfli4l13/src/parser/states/assignmentstates/AbstractAssignmentState.java

parser.states.definitionstates.AbstractDefinitionState Klassenreferenz

parser.states.definitionstates.AbstractDefinitionStateKlassendiagramm für parser.states.definitionstates.AbstractDefinitionState:



Zusammengehörigkeiten von parser.states.definitionstates.AbstractDefinitionState:



Öffentliche Methoden

- abstract void parse (TempVariableDefinition tVD, Buffer< AbstractSymbol > buffer, DefinitionFileParser dFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Sandra

Dokumentation der Elementfunktionen

abstract void parser.states.definitionstates.AbstractDefinitionState.parse (TempVariableDefinition *tVD*, Buffer< AbstractSymbol > *buffer*, DefinitionFileParser *dFP*) throws InterruptedException, ParserException `[pure virtual]`

Parameter:

tVD	TempVariableDefinition
dFP	DefinitionFileParser

buffer	Buffer<Symbol>

Rückgabe:

Ausnahmebehandlung:

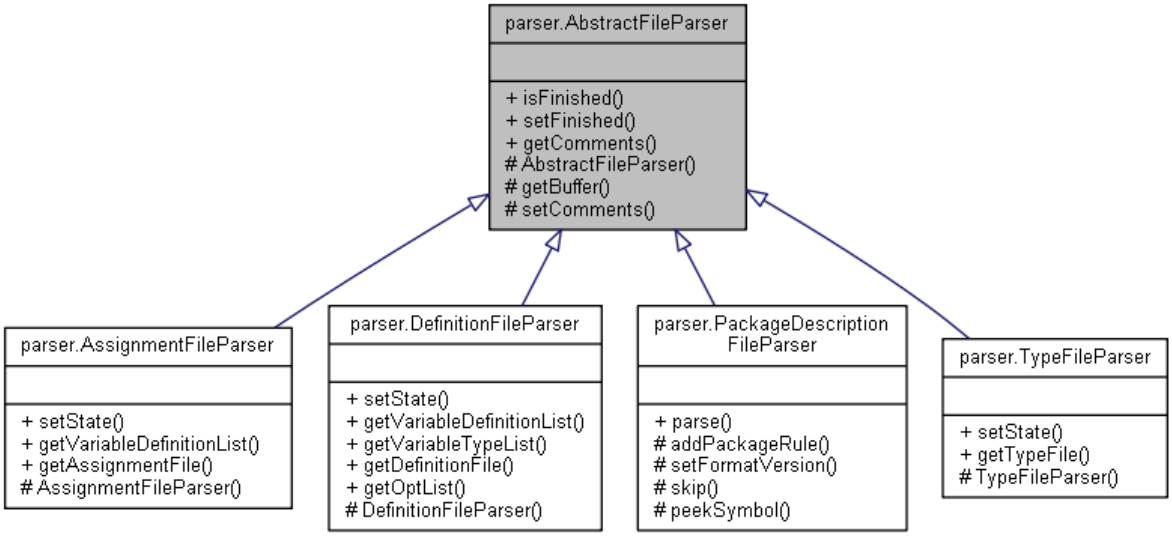
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

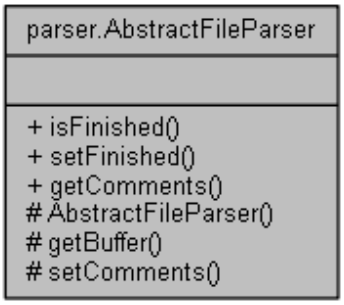
- workspace/ipfli4l13/src/parser/states/definitionstates/AbstractDefinitionState.java

parser.AbstractFileParser Klassenreferenz

parser.AbstractFileParserKlassendiagramm für parser.AbstractFileParser:



Zusammengehörigkeiten von parser.AbstractFileParser:



Öffentliche Methoden

- Boolean isFinished ()
- void setFinished (final Boolean finished)
- List< Comment > getComments ()

Geschützte Methoden

- AbstractFileParser (final Buffer< AbstractSymbol > symbolBuffer)
- Buffer< AbstractSymbol > getBuffer ()
- void setComments (final List< Comment > comments)

Ausführliche Beschreibung

Autor:
hfw410wi - Mark Wittig

Beschreibung der Konstruktoren und Destruktoren

parser.AbstractFileParser.AbstractFileParser (final Buffer< AbstractSymbol > *symbolBuffer*)[protected]

constructor for create the PackageParser.

Parameter:

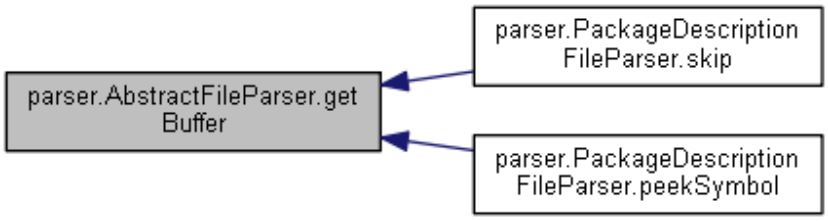
symbolBuffer	Buffer<AbstractSymbol>

Dokumentation der Elementfunktionen

Buffer<AbstractSymbol> parser.AbstractFileParser.getBuffer ()[protected]

Rückgabe:
the buffer

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



List<Comment> parser.AbstractFileParser.getComments ()

Rückgabe:
the comments

Boolean parser.AbstractFileParser.isFinished ()

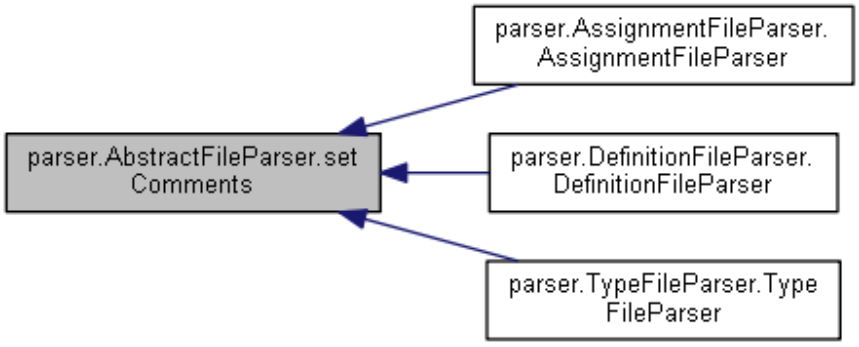
Rückgabe:
the finished

void parser.AbstractFileParser.setComments (final List< Comment > *comments*)[protected]

Parameter:

comments	the comments to set

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

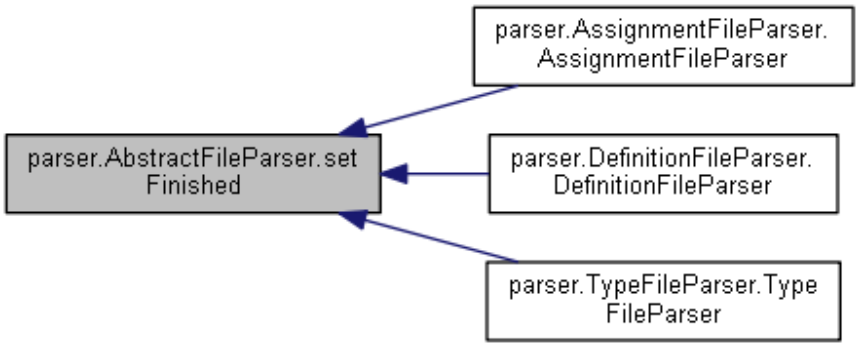


void parser.AbstractFileParser.setFinished (final Boolean *finished*)

Parameter:

finished	the finished to set

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

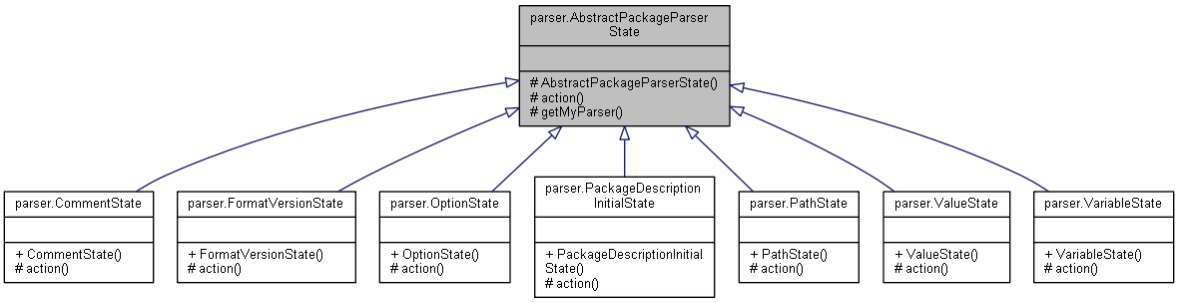


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

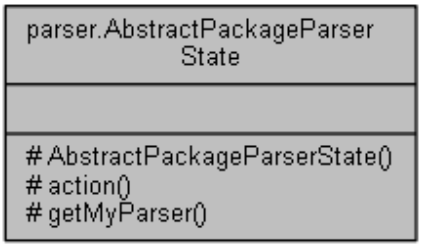
- workspace/ipfli4l13/src/parser/AbstractFileParser.java

parser.AbstractPackageParserState Klassenreferenz

parser.AbstractPackageParserStateKlassendiagramm für parser.AbstractPackageParserState:



Zusammengehörigkeiten von parser.AbstractPackageParserState:



Geschützte Methoden

- AbstractPackageParserState (final PackageDescriptionFileParser parser)
- abstract AbstractPackageParserState action (PackageDescriptionFileSymbol symbol, Map< String, AbstractVariableDefinition > variableDef) throws InterruptedException, ParserException
- PackageDescriptionFileParser getMyParser ()

Ausführliche Beschreibung

The State for the PackageDescriptionFileScanner.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

parser.AbstractPackageParserState.AbstractPackageParserState (final PackageDescriptionFileParser *parser*)`[protected]`

the Konstruktor for the AbstractParserState.

Parameter:

parser	the parser, which owns the state.

Dokumentation der Elementfunktionen

abstract AbstractPackageParserState parser.AbstractPackageParserState.action (PackageDescriptionFileSymbol *symbol*, Map< String, AbstractVariableDefinition > *variableDef*) throws InterruptedException, ParserException`[protected]`, `[pure virtual]`

handle the given Symbol.

Parameter:

symbol	is the given Symbol.
variableDef	is the List of variableDefinition, which the Assignmentparser has created.

Ausnahmebehandlung:

InterruptedException	from the buffer.
ParserException	if there is any Problem while parsing

Rückgabe:

AbstractParserState, which is the actual State of the Parser.

PackageDescriptionFileParser parser.AbstractPackageParserState.getMyParser ()`[protected]`

Rückgabe:

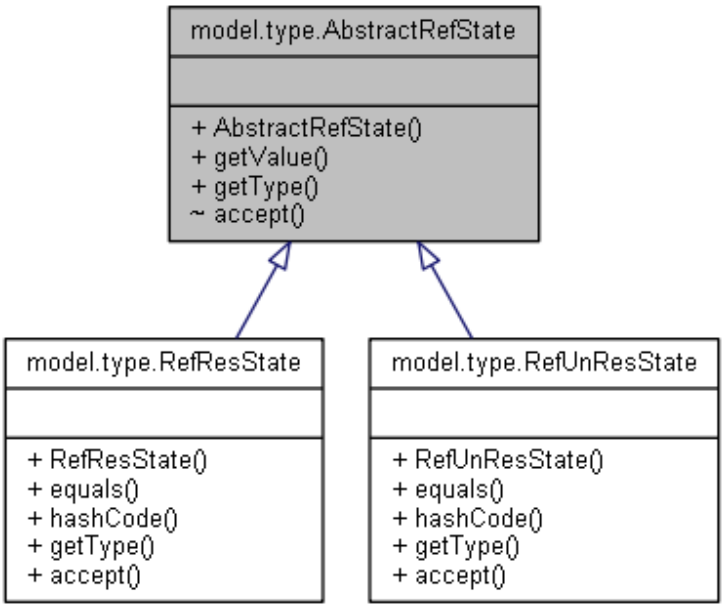
the Parser;

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

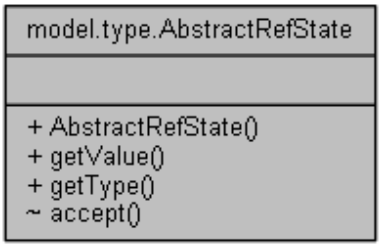
- workspace/ipfli4l13/src/parser/AbstractPackageParserState.java

model.type.AbstractRefState Klassenreferenz

model.type.AbstractRefStateKlassendiagramm für model.type.AbstractRefState:



Zusammengehörigkeiten von model.type.AbstractRefState:



Öffentliche Methoden

- AbstractRefState (final String value)
- String getValue ()
- abstract NamedVariableType getType () throws ReferenceIsUnresolvedException

---

Ausführliche Beschreibung

Representation for the state of a RegExRefpart.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.AbstractRefState.AbstractRefState (final String *value*)

Constructor for an AbstractRefState, just sets the field value.

Parameter:

value	The value to set.

Dokumentation der Elementfunktionen

abstract NamedVariableType model.type.AbstractRefState.getType () throws ReferenceIsUnresolvedException [pure virtual]

Returns the field type, if it is an RefResState, else an exception is thrown.

Rückgabe:

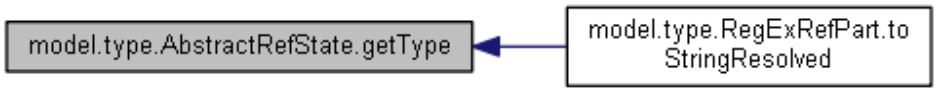
this.type

Ausnahmebehandlung:

ReferenceIsUnresolvedException	If the state isnt a RefResState.

Implementiert in model.type.RefResState und model.type.RefUnResState.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



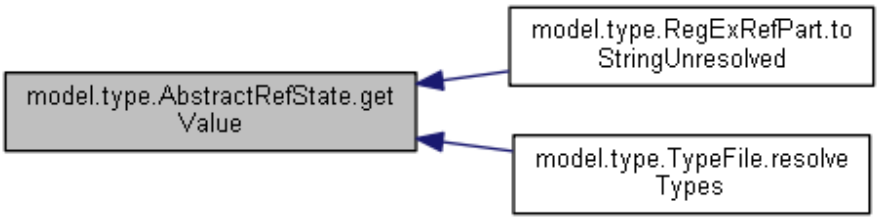
String model.type.AbstractRefState.getValue ()

Returns the field value. No side effects.

Rückgabe:

Return this.value()

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

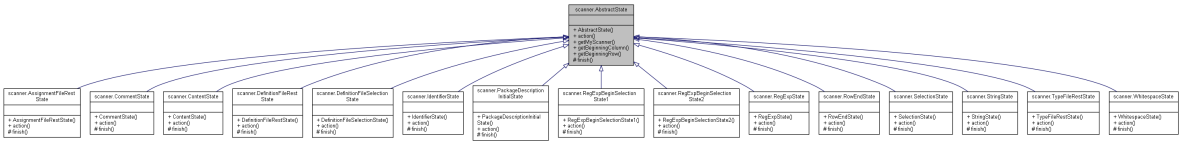


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

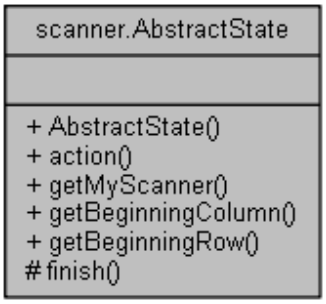
- workspace/ipfli4l13/src/model/type/AbstractRefState.java

scanner.AbstractState Klassenreferenz

scanner.AbstractStateKlassendiagramm für scanner.AbstractState:



Zusammengehörigkeiten von scanner.AbstractState:



Öffentliche Methoden

- AbstractState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)
- abstract AbstractState action (Character character, Buffer< AbstractSymbol > currentResult, String dataPath) throws InterruptedException
- Scanner getMyScanner ()
- Integer getBeginningColumn ()
- Integer getBeginningRow ()

Geschützte Methoden

- abstract void finish (Buffer< AbstractSymbol > currentResult, String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

This class represents the states as abstract class.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.AbstractState.AbstractState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

This constructor create a new State.

Parameter:

myScanner	is the used scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the begin row of the symbol

Dokumentation der Elementfunktionen

abstract AbstractState scanner.AbstractState.action (Character *character*, Buffer< AbstractSymbol > *currentResult*, String *dataPath*) throws InterruptedException `[pure virtual]`

This method represents the action which will be executed, if it reached.

Parameter:

character	is the used Character.
currentResult	is the Buffer of symbols.
dataPath	is the path to the dataStream

Ausnahmebehandlung:

InterruptedException	for a Interruption in the method

Rückgabe:

the actual state of the scanner.

abstract void scanner.AbstractState.finish (Buffer< AbstractSymbol > *currentResult*, String *dataPath*) throws InterruptedException `[protected]`, `[pure virtual]`

Method, which will be call at the end of each row.

Parameter:

currentResult	is the buffer of symbols.
dataPath	is the path.

Ausnahmebehandlung:

InterruptedException	from the Buffer.

Integer scanner.AbstractState.getBeginningColumn ()

Rückgabe:
the begincolumn of the Chracter in the stream.

Integer scanner.AbstractState.getBeginningRow ()

Rückgabe:
the beginning row.

Scanner scanner.AbstractState.getMyScanner ()

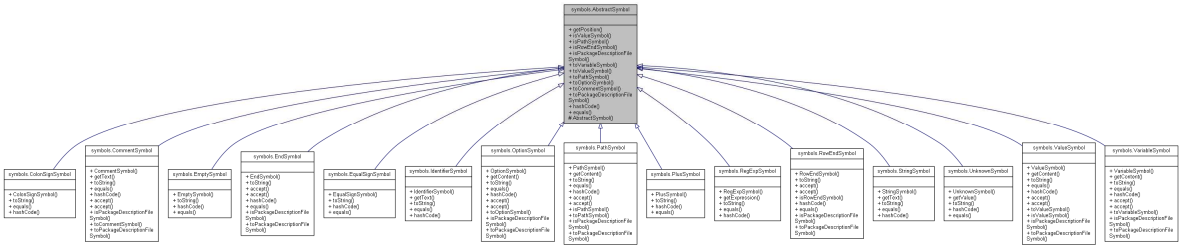
Rückgabe:
the used scanner.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

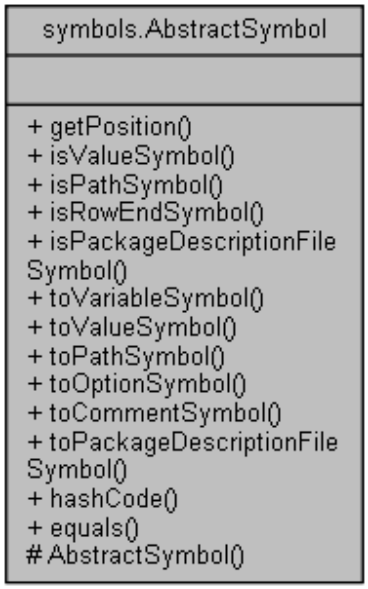
- workspace/ipfli4l13/src/scanner/AbstractState.java

symbols.AbstractSymbol Klassenreferenz

symbols.AbstractSymbolKlassendiagramm für symbols.AbstractSymbol:



Zusammengehörigkeiten von symbols.AbstractSymbol:



Öffentliche Methoden

- Position getPosition ()
- Boolean isValueSymbol ()
- Boolean isPathSymbol ()
- Boolean isRowEndSymbol ()
- Boolean isPackageDescriptionFileSymbol ()
- VariableSymbol toVariableSymbol ()
- ValueSymbol toValueSymbol ()
- PathSymbol toPathSymbol ()
- OptionSymbol toOptionSymbol ()
- CommentSymbol toCommentSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()
- int **hashCode** ()
- boolean **equals** (final Object obj)

Geschützte Methoden

- AbstractSymbol (final Position position)

Ausführliche Beschreibung

This abstract class represents the symbols.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

symbols.AbstractSymbol.AbstractSymbol (final Position *position*)[protected]

This constructor instantiate the Symbol.

Parameter:

position	(row and column) of the data in the datastream. The position of the first sign of the symbol inclusive of the # or ' etc.

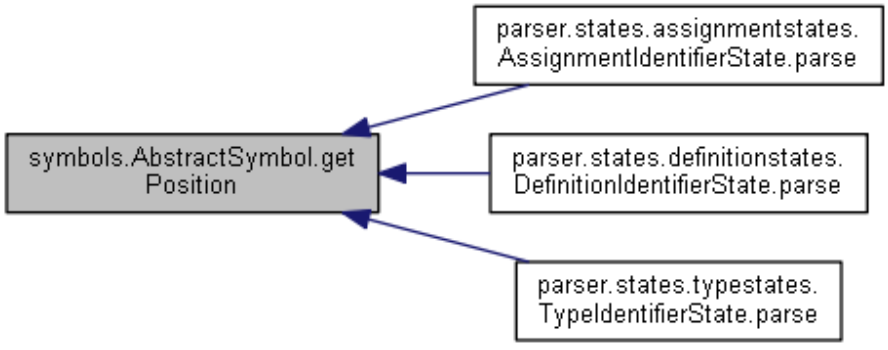
Dokumentation der Elementfunktionen

Position symbols.AbstractSymbol.getPosition ()

Rückgabe:

the position

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

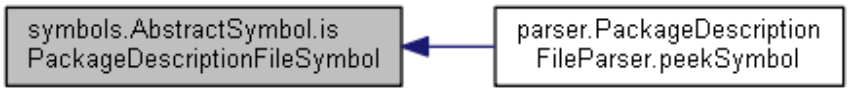


Boolean symbols.AbstractSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



Boolean symbols.AbstractSymbol.isPathSymbol ()

Rückgabe:

true, if the Symbol is a path symbol.

Boolean symbols.AbstractSymbol.isRowEndSymbol ()

Rückgabe:

true, if the Symbol is a row end symbol.

Boolean symbols.AbstractSymbol.isValueSymbol ()

Rückgabe:

true, if the Symbol is a value symbol.

CommentSymbol symbols.AbstractSymbol.toCommentSymbol ()

Rückgabe:

an error, if the symbol is not a comment symbol

OptionSymbol symbols.AbstractSymbol.toOptionSymbol ()

Rückgabe:

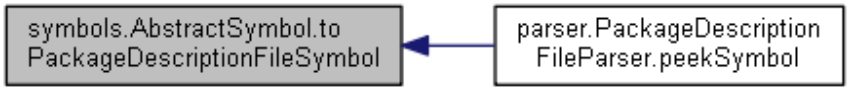
an error, if the symbol is not an option symbol

PackageDescriptionFileSymbol symbols.AbstractSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:

a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



PathSymbol symbols.AbstractSymbol.toPathSymbol ()

Rückgabe:
an error, if the symbol is not a path symbol

ValueSymbol symbols.AbstractSymbol.toValueSymbol ()

Rückgabe:
an error, if the symbol is not a value symbol

VariableSymbol symbols.AbstractSymbol.toVariableSymbol ()

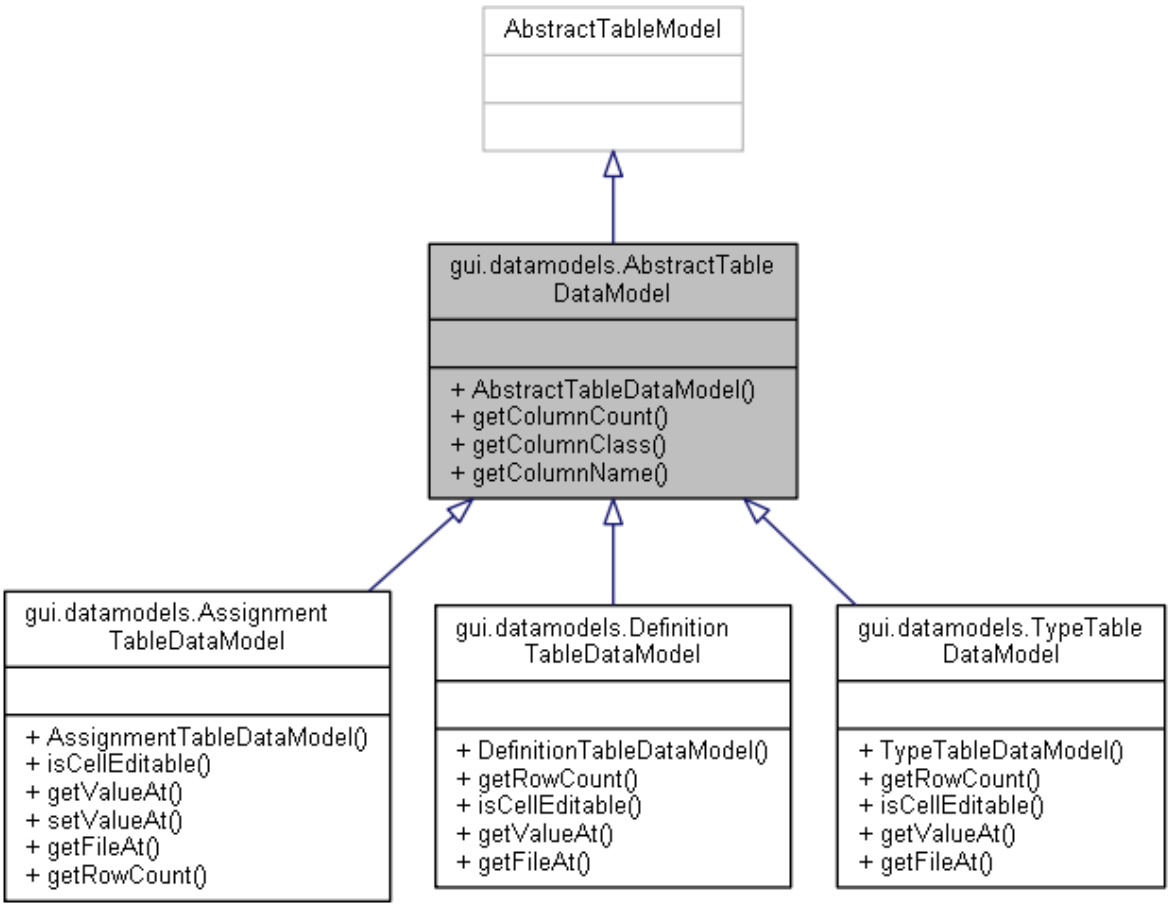
Rückgabe:
an error, if the symbol is not a variable symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

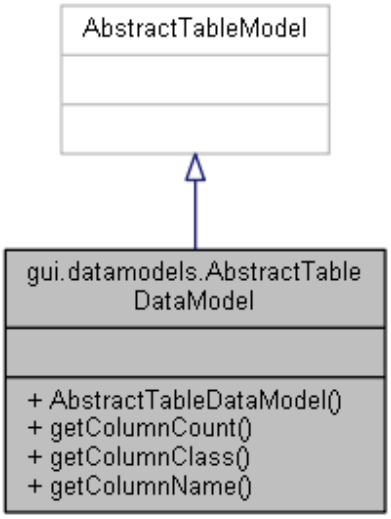
- workspace/ipfli4l13/src/symbols/AbstractSymbol.java

gui.datamodels.AbstractTableDataModel Klassenreferenz

gui.datamodels.AbstractTableDataModelKlassendiagramm für gui.datamodels.AbstractTableDataModel:



Zusammengehörigkeiten von gui.datamodels.AbstractTableDataModel:



Öffentliche Methoden

- AbstractTableDataModel (final String[] headers)
- int **getColumnCount** ()
- Class< String > **getColumnClass** (final int c)
- String **getColumnName** (final int c)

Ausführliche Beschreibung

Our own datamodel for tables.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.datamodels.AbstractTableDataModel.AbstractTableDataModel (final String[] *headers*)

Parameter:

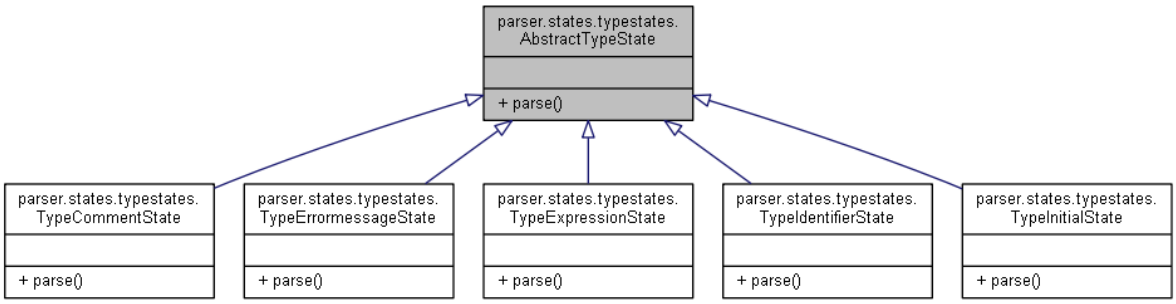
headers	the headers.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

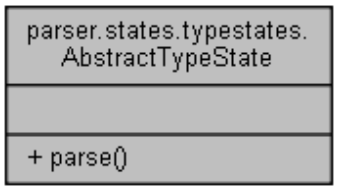
- workspace/ipfli4l13/src/gui/datamodels/AbstractTableDataModel.java

parser.states.typestates.AbstractTypeState Klassenreferenz

parser.states.typestates.AbstractTypeStateKlassendiagramm für parser.states.typestates.AbstractTypeState:



Zusammengehörigkeiten von parser.states.typestates.AbstractTypeState:



Öffentliche Methoden

- abstract void parse (TempVariableType tVT, Buffer< AbstractSymbol > buffer, TypeFileParser tFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Sandra

Dokumentation der Elementfunktionen

abstract void parser.states.typestates.AbstractTypeState.parse (TempVariableType *tVT*, Buffer< AbstractSymbol > *buffer*, TypeFileParser *tFP*) throws InterruptedException, ParserException `[pure virtual]`

Parameter:

tVT	tempVariableType

buffer	buffer
tFP	typeFileParser

Rückgabe:

Ausnahmebehandlung:

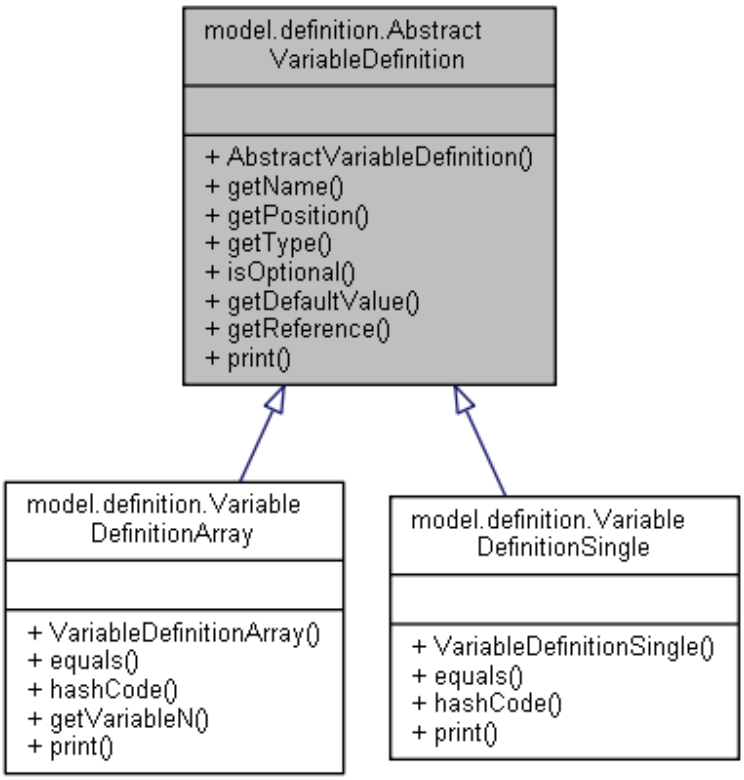
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

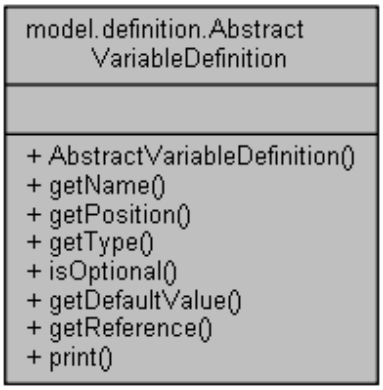
- workspace/ipfli4l13/src/parser/states/typestates/AbstractTypeState.java

model.definition.AbstractVariableDefinition Klassenreferenz

model.definition.AbstractVariableDefinitionKlassendiagramm für model.definition.AbstractVariableDefinition:



Zusammengehörigkeiten von model.definition.AbstractVariableDefinition:



Öffentliche Methoden

- AbstractVariableDefinition (final String name, final AbstractVariableType type, final boolean isOptional, final DefaultValue defaultValue, final VariableOptReference reference, final Position position)

- String getName ()
- Position getPosition ()
- AbstractVariableType getType ()
- boolean isOptional ()
- DefaultValue getDefaultValue ()
- VariableOptReference getReference ()
- abstract String print ()

Ausführliche Beschreibung

VariableDefinition represents an entry in the "\check\<package>.txt" files.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

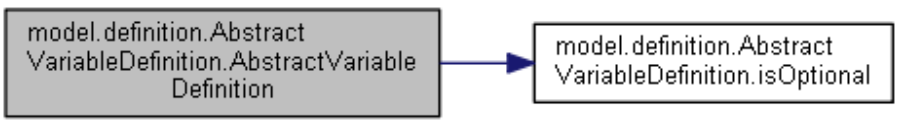
model.definition.AbstractVariableDefinition.AbstractVariableDefinition (final String *name*, final AbstractVariableType *type*, final boolean *isOptional*, final DefaultValue *defaultValue*, final VariableOptReference *reference*, final Position *position*)

Constructor, just sets the fields.

Parameter:

name	The name of the definition
type	The type of the definition
isOptional	Represents if the definition is optional
defaultValue	The default value of the definition.
reference	Represents if the definition has an opt or not.
position	The row to set.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Dokumentation der Elementfunktionen

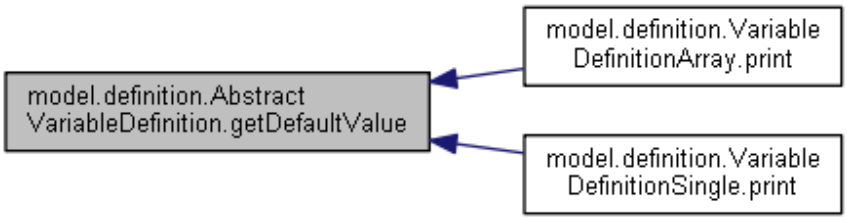
DefaultValue model.definition.AbstractVariableDefinition.getDefaultValue ()

Returns the field defaultValue. No side effects.

Rückgabe:

this.defaultValue

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



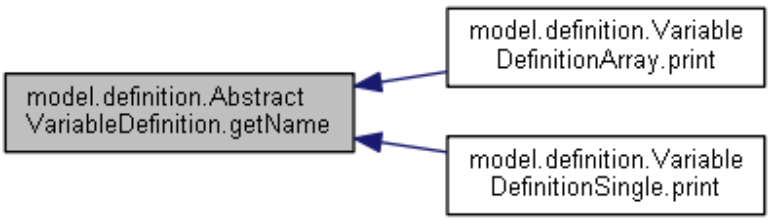
String model.definition.AbstractVariableDefinition.getName ()

Returns the field name. No side effects.

Rückgabe:

this.name

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



Position model.definition.AbstractVariableDefinition.getPosition ()

Returns the field row. No side effects.

Rückgabe:

this.row

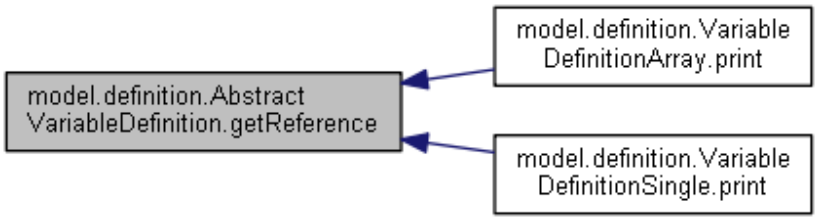
VariableOptReference model.definition.AbstractVariableDefinition.getReference ()

Returns the field reference. No side effects.

Rückgabe:

this.reference

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



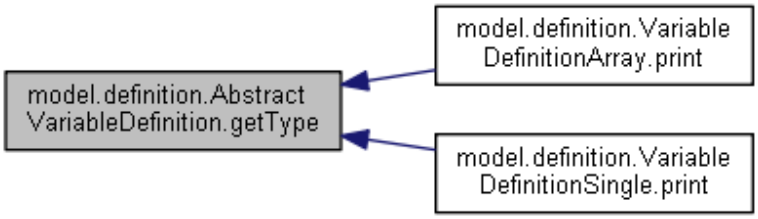
AbstractVariableType model.definition.AbstractVariableDefinition.getType ()

Returns the field type. No side effects.

Rückgabe:

this.type

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



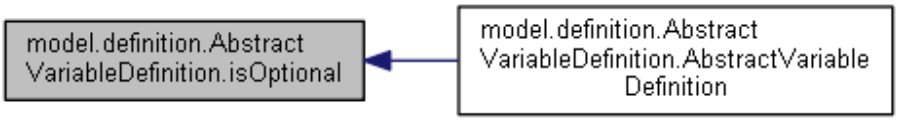
boolean model.definition.AbstractVariableDefinition.isOptional ()

Returns the field isOptional. No side effects.

Rückgabe:

this.isOptional

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



abstract String model.definition.AbstractVariableDefinition.print ()[pure virtual]

Returns the line to print.

Rückgabe:

Returns a string value, representing a line to print.

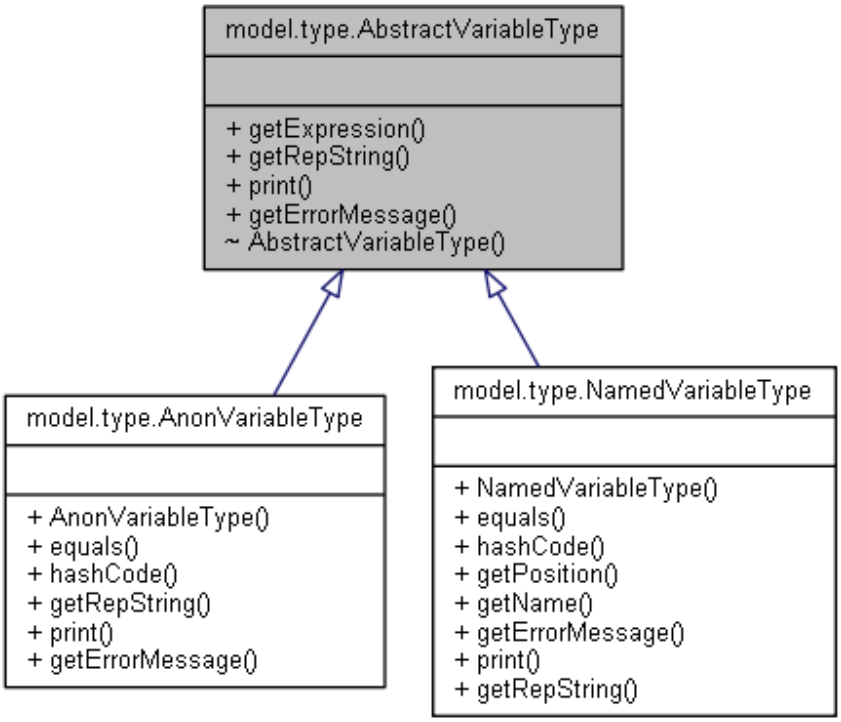
Implementiert in model.definition.VariableDefinitionArray und model.definition.VariableDefinitionSingle.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

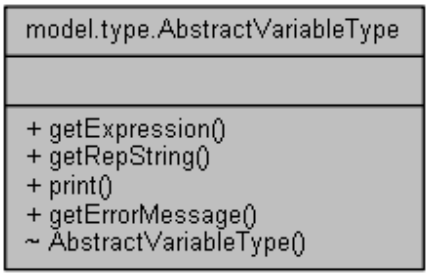
- workspace/ipfli4l13/src/model/definition/AbstractVariableDefinition.java

model.type.AbstractVariableType Klassenreferenz

model.type.AbstractVariableTypeKlassendiagramm für model.type.AbstractVariableType:



Zusammengehörigkeiten von model.type.AbstractVariableType:



Öffentliche Methoden

- RegEx getExpression ()
- abstract String getRepString ()
- abstract String print ()
- abstract String getErrorMessage ()

Ausführliche Beschreibung

AbstractVariableType.

Autor:

Phil

Dokumentation der Elementfunktionen

abstract String model.type.AbstractVariableType.getErrorMessage ()[pure virtual]

Return the field errorMessage. No side effects.

Rückgabe:

this.errorMessage

Implementiert in model.type.NamedVariableType und model.type.AnonVariableType.

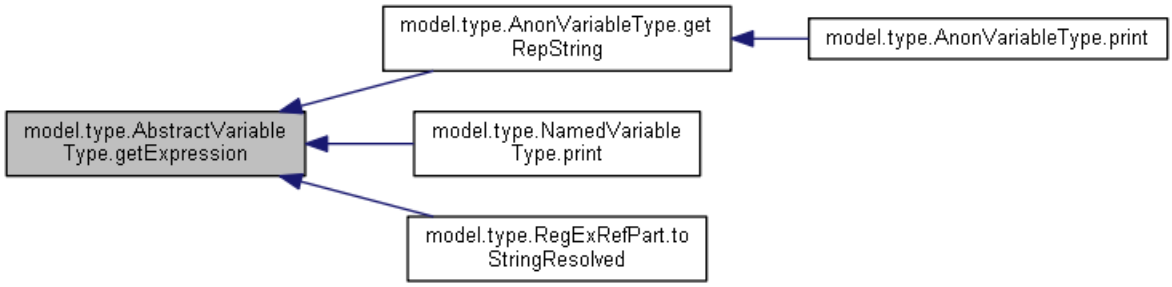
RegEx model.type.AbstractVariableType.getExpression ()

Return the field expression. No side effects.

Rückgabe:

this.expression

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



abstract String model.type.AbstractVariableType.getRepString ()[pure virtual]

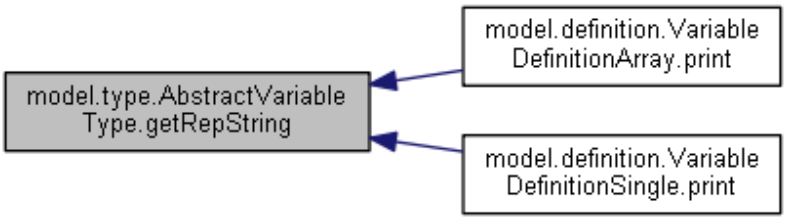
Returns the type-representation for printing.

Rückgabe:

Typerepresentation for printing.

Implementiert in model.type.NamedVariableType und model.type.AnonVariableType.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



abstract String model.type.AbstractVariableType.print ()[pure virtual]

Returns the line to print.

Rückgabe:

Returns a string value, representing a line to print.

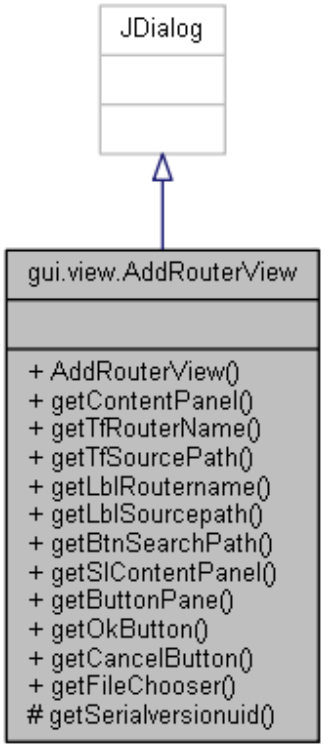
Implementiert in model.type.NamedVariableType und model.type.AnonVariableType.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

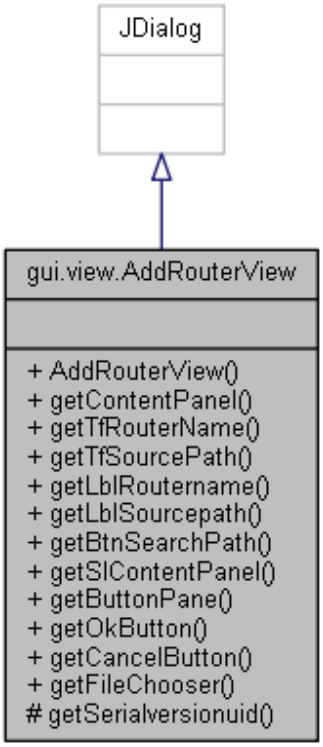
- workspace/ipfli4l13/src/model/type/AbstractVariableType.java

gui.view.AddRouterView Klassenreferenz

gui.view.AddRouterViewKlassendiagramm für gui.view.AddRouterView:



Zusammengehörigkeiten von gui.view.AddRouterView:



Öffentliche Methoden

- AddRouterView ()
- JPanel getContentPanel ()
- JTextField getTfRouterName ()
- JTextField getTfSourcePath ()
- JLabel getLblRoutername ()
- JLabel getLblSourcepath ()
- JButton getBtnSearchPath ()
- SpringLayout getSlContentPanel ()
- JPanel getButtonPane ()
- JButton getOkButton ()
- JButton getCancelButton ()
- JFileChooser getFileChooser ()

Geschützte, statische Methoden

- static long getSerialversionuid ()

Ausführliche Beschreibung

The dialog box for adding a router.

Autor:

Mark Wittig / Lukas Aliger

Beschreibung der Konstruktoren und Destruktoren

gui.view.AddRouterView.AddRouterView ()

Create the dialog.

Dokumentation der Elementfunktionen

JButton gui.view.AddRouterView.getBtnSearchPath ()

Rückgabe:
the btnSearchPath

JPanel gui.view.AddRouterView.getButtonPane ()

Rückgabe:
the buttonPane

JButton gui.view.AddRouterView.getCancelButton ()

Rückgabe:
the cancelButton

JPanel gui.view.AddRouterView.getContentPanel ()

Rückgabe:
the contentPanel

JFileChooser gui.view.AddRouterView.getFileChooser ()

Rückgabe:
the fileChooser

JLabel gui.view.AddRouterView.getLblRoutername ()

Rückgabe:
the lblRoutername

JLabel gui.view.AddRouterView.getLblSourcepath ()

Rückgabe:
the lblSourcepath

JButton gui.view.AddRouterView.getOkButton ()

Rückgabe:
the okButton

static long gui.view.AddRouterView.getSerialversionuid ()[static], [protected]

Rückgabe:
the serialversionuid

SpringLayout gui.view.AddRouterView.getSlContentPanel ()

Rückgabe:
the slContentPanel

JTextField gui.view.AddRouterView.getTfRouterName ()

Rückgabe:
the tfRouterName

JTextField gui.view.AddRouterView.getTfSourcePath ()

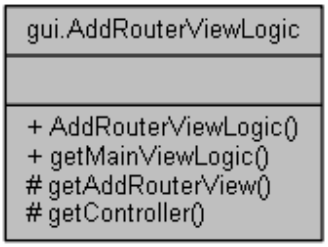
Rückgabe:
the tfSourcePath

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/view/AddRouterView.java

gui.AddRouterViewLogic Klassenreferenz

gui.AddRouterViewLogicZusammengehörigkeiten von gui.AddRouterViewLogic:



Öffentliche Methoden

- AddRouterViewLogic (final MainController controller, final MainViewLogic mainViewLogic)
- MainViewLogic getMainViewLogic ()

Geschützte Methoden

- AddRouterView getAddRouterView ()
- MainController getController ()

Ausführliche Beschreibung

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.AddRouterViewLogic.AddRouterViewLogic (final MainController *controller*, final MainViewLogic *mainViewLogic*)

constructor. Creates the view and sets fields.

Parameter:

mainViewLogic	The main view logic to set.
controller	the controller to set.

Dokumentation der Elementfunktionen

AddRouterView gui.AddRouterViewLogic.getAddRouterView ()[protected]

returns this.addRouterView.

Rückgabe:
this.addRouterView.

MainController gui.AddRouterViewLogic.getController ()[protected]

returns this.controller.

Rückgabe:
this.controller.

MainViewLogic gui.AddRouterViewLogic.getMainViewLogic ()

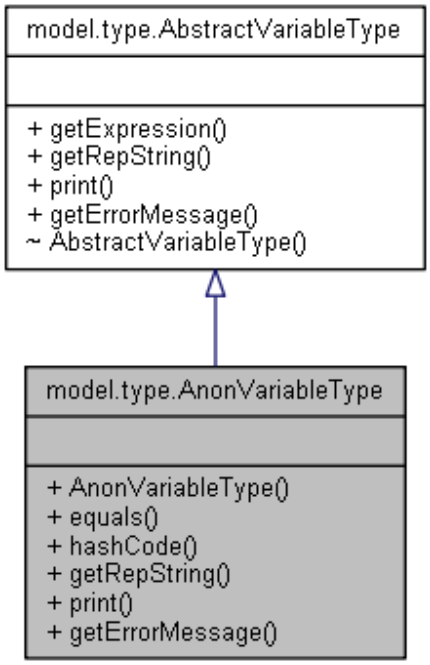
Rückgabe:
the mainViewLogic

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

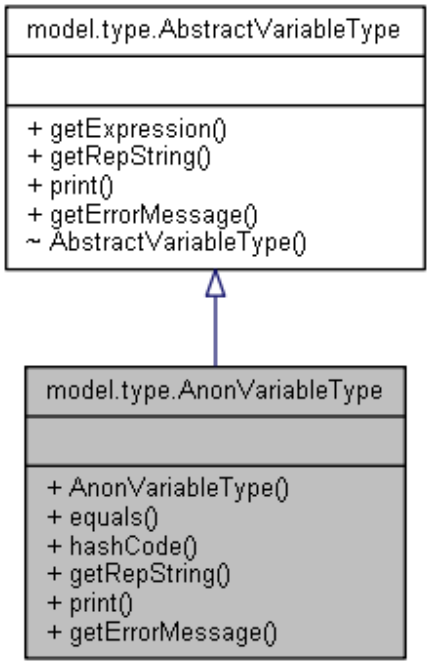
- workspace/ipfli4l13/src/gui/AddRouterViewLogic.java

model.type.AnonVariableType Klassenreferenz

model.type.AnonVariableTypeKlassendiagramm für model.type.AnonVariableType:



Zusammengehörigkeiten von model.type.AnonVariableType:



Öffentliche Methoden

- AnonVariableType (final RegEx expression)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- String getRepString ()
- String print ()
- String getErrorMessage ()

Ausführliche Beschreibung

Autor:
Phil

Beschreibung der Konstruktoren und Destruktoren

model.type.AnonVariableType.AnonVariableType (final RegEx *expression*)

Constructor for AnonVariableType, just calls super(expression).

Parameter:

expression	The expression to set.

Dokumentation der Elementfunktionen

String model.type.AnonVariableType.getErrorMessage ()[virtual]

Return the field errorMessage. No side effects.

Rückgabe:
this.errorMessage

Implementiert model.type.AbstractVariableType.

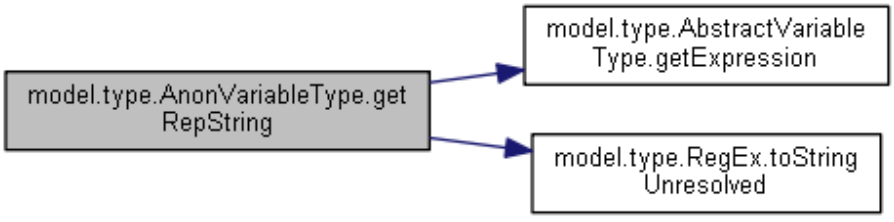
String model.type.AnonVariableType.getRepString ()[virtual]

Returns the type-representation for printing.

Rückgabe:
Typerepresentation for printing.

Implementiert model.type.AbstractVariableType.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

String model.type.AnonVariableType.print ()[virtual]

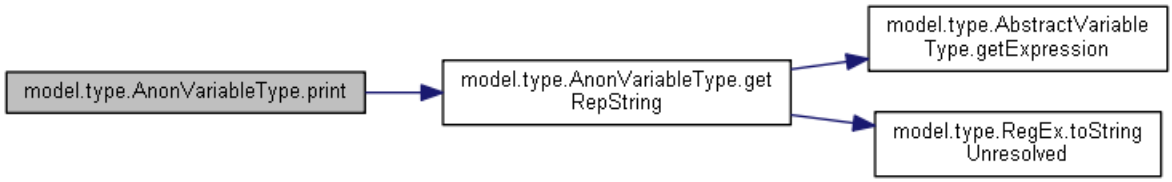
Returns the line to print.

Rückgabe:

Returns a string value, representing a line to print.

Implementiert model.type.AbstractVariableType.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

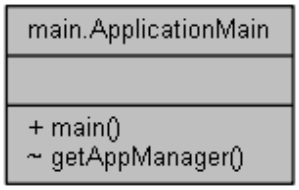


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/type/AnonVariableType.java

main.ApplicationMain Klassenreferenz

main.ApplicationMainZusammengehörigkeiten von main.ApplicationMain:



Öffentliche, statische Methoden

- static void main (final String[] args)

Ausführliche Beschreibung

Autor:

Phil The real entry point.

Dokumentation der Elementfunktionen

static void main.ApplicationMain.main (final String[] *args*)`[static]`

this is a real main-method (not a debugger-leftover).

Parameter:

args	The args.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/main/ApplicationMain.java

main.ApplicationManager Klassenreferenz

main.ApplicationManagerZusammengehörigkeiten von main.ApplicationManager:

Ausführliche Beschreibung

Starts and prepares the application.

Autor:

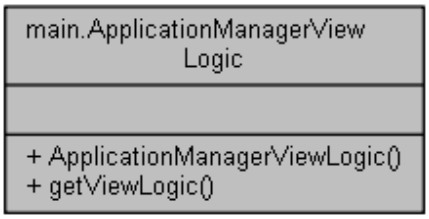
Phil

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/main/ApplicationManager.java

main.ApplicationManagerViewLogic Klassenreferenz

main.ApplicationManagerViewLogicZusammengehörigkeiten von main.ApplicationManagerViewLogic:



Öffentliche Methoden

- ApplicationManagerViewLogic (final MainController controller)
- MainViewLogic getViewLogic ()

Ausführliche Beschreibung

Applicationmanager for viewlogic-part.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

main.ApplicationManagerViewLogic.ApplicationManagerViewLogic (final MainController *controller*)

Parameter:

controller	The controller to set.

Dokumentation der Elementfunktionen

MainViewLogic main.ApplicationManagerViewLogic.getViewLogic ()

Rückgabe:

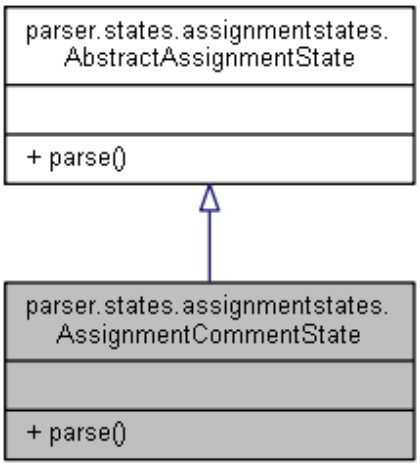
the viewLogic

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

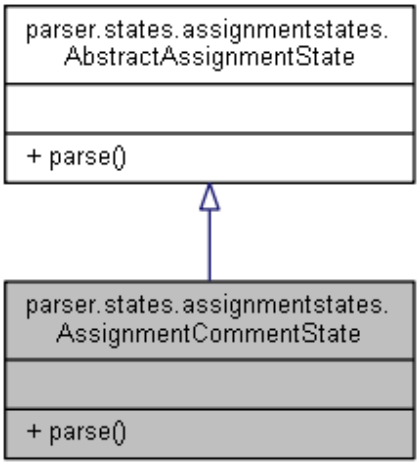
- workspace/ipfli4l13/src/main/ApplicationManagerViewLogic.java

parser.states.assignmentstates.AssignmentCommentState Klassenreferenz

parser.states.assignmentstates.AssignmentCommentStateKlassendiagramm für parser.states.assignmentstates.AssignmentCommentState:



Zusammengehörigkeiten von parser.states.assignmentstates.AssignmentCommentState:



Öffentliche Methoden

- void parse (final TempVariableAssignment tVA, final Buffer< AbstractSymbol > buffer, final AssignmentFileParser aFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.assignmentstates.AssignmentCommentState.parse (final TempVariableAssignment *tVA*, final Buffer< AbstractSymbol > *buffer*, final AssignmentFileParser *aFP*) throws InterruptedException, ParserException

Parameter:

tVA	TempVariableAssignment
buffer	Buffer<Symbol>
aFP	AssignmentFileParser

Ausnahmebehandlung:

InterruptedException	InterruptedException
ParserException	ParserException

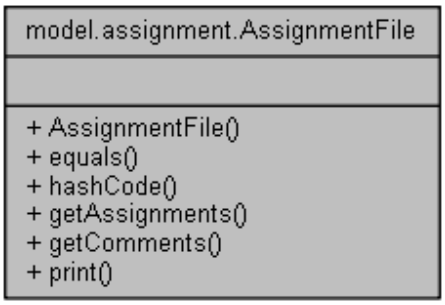
Rückgabe:

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/states/assignmentstates/AssignmentCommentState.java

model.assignment.AssignmentFile Klassenreferenz

model.assignment.AssignmentFileZusammengehörigkeiten von model.assignment.AssignmentFile:



Öffentliche Methoden

- AssignmentFile (final Map< String, VariableAssignment > assignments2, final List< Comment > comments)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- java.util.Map< String,
- VariableAssignment > getAssignments ()
- java.util.List< Comment > getComments ()
- String print (final String pathToPrint) throws printer.PrinterException

---

Ausführliche Beschreibung

AssignmentFile represents the "\config\<package>.txt" files.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.assignment.AssignmentFile.AssignmentFile (final Map< String, VariableAssignment > *assignments2*, final List< Comment > *comments*)

Constructor for AssignmentFile, just sets the field.

Parameter:

assignments2	The assignments to set.
comments	The comments to set.

Dokumentation der Elementfunktionen

java.util.Map<String, VariableAssignment> model.assignment.AssignmentFile.getAssignments ()

Returns the field assignments. No side effects.

Rückgabe:

this.assignments;

java.util.List<Comment> model.assignment.AssignmentFile.getComments ()

Rückgabe:

the comments

String model.assignment.AssignmentFile.print (final String *pathToPrint*) throws printer.PrinterException

Prints an AssignmentFile containing the VariableAssignments and Comments.

Parameter:

pathToPrint	The path where to print the file.

Ausnahmebehandlung:

printer.PrinterException	If a IOException is thrown.
printer.PrinterException	

Rückgabe:

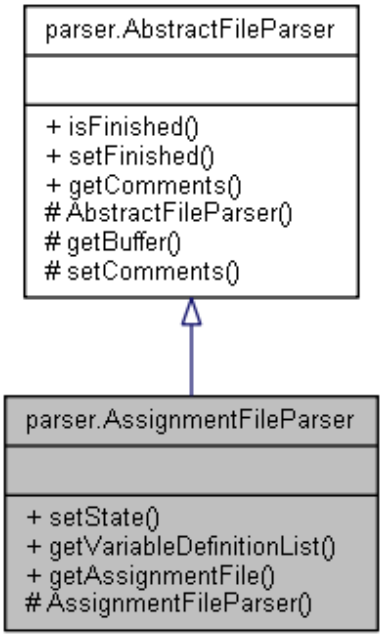
Returns a string representing the printed lines (for testing).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

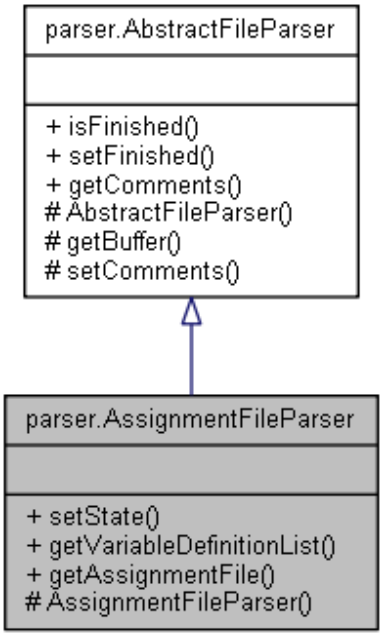
- workspace/ipfli4l13/src/model/assignment/AssignmentFile.java

parser.AssignmentFileParser Klassenreferenz

parser.AssignmentFileParserKlassendiagramm für parser.AssignmentFileParser:



Zusammengehörigkeiten von parser.AssignmentFileParser:



Öffentliche Methoden

- void setState (final AbstractAssignmentState state)
- Map< String,
- AbstractVariableDefinition > getVariableDefinitionList ()
- AssignmentFile getAssignmentFile ()

Geschützte Methoden

- AssignmentFileParser (final Buffer< AbstractSymbol > buffer, final Map< String, AbstractVariableDefinition > varDefList) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
hfw410wi - Mark Wittig

Beschreibung der Konstruktoren und Destruktoren

parser.AssignmentFileParser.AssignmentFileParser (final Buffer< AbstractSymbol > *buffer*, final Map< String, AbstractVariableDefinition > *varDefList*) throws InterruptedException, ParserException `[protected]`

The AssignmentFileParser will automatically generate a AssignmentFile after instantiation which can be obtained via getAssignmentFile().

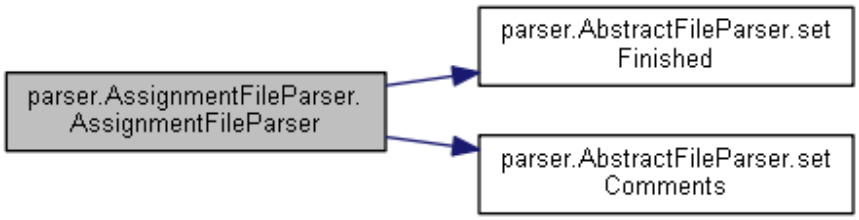
Parameter:

buffer	Buffer<Symbol>
varDefList	variableDefinitionList

Ausnahmebehandlung:

ParserException	ParserException
InterruptedException	InterruptedException

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Dokumentation der Elementfunktionen

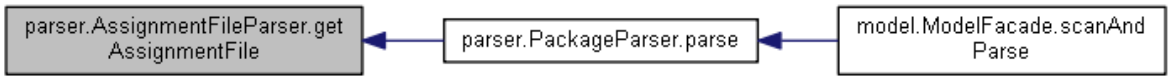
AssignmentFile parser.AssignmentFileParser.getAssignmentFile ()

Returns the desired AssignmentFile which is automatically created right after instantiation.

Rückgabe:

the assignmentFile

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

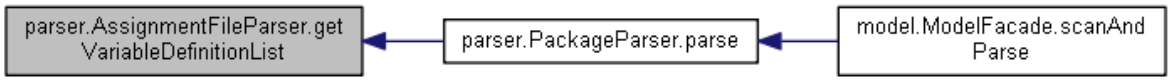


Map<String, AbstractVariableDefinition> parser.AssignmentFileParser.getVariableDefinitionList ()

Rückgabe:

the variableDefinitionList

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



void parser.AssignmentFileParser.setState (final AbstractAssignmentState *state*)

Parameter:

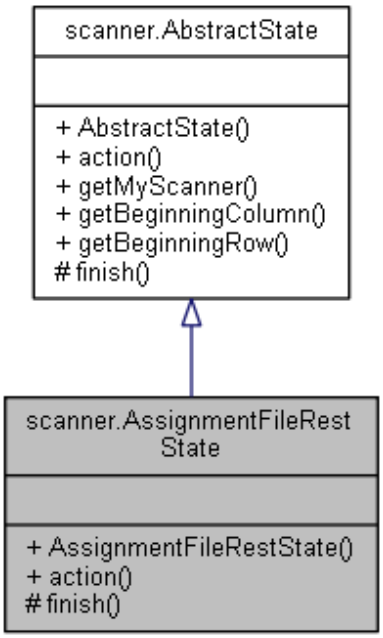
state	the state to set

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

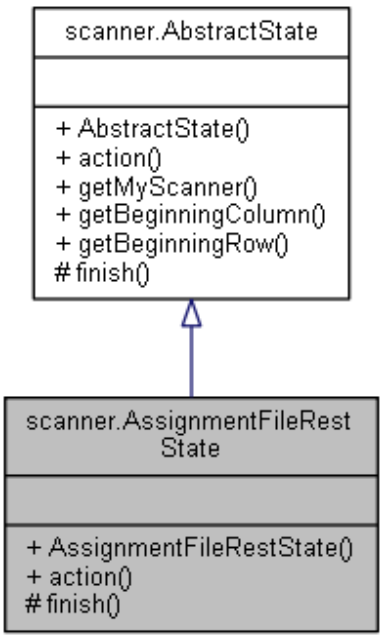
- workspace/ipfli4l13/src/parser/AssignmentFileParser.java

scanner.AssignmentFileRestState Klassenreferenz

scanner.AssignmentFileRestStateKlassendiagramm für scanner.AssignmentFileRestState:



Zusammengehörigkeiten von scanner.AssignmentFileRestState:



Öffentliche Methoden

- AssignmentFileRestState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)
- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath)

Ausführliche Beschreibung

This class represent the AssignmentFileSelectionState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.AssignmentFileRestState.AssignmentFileRestState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

This is a the constructor for a new AssignmentFileSelectionState.

Parameter:

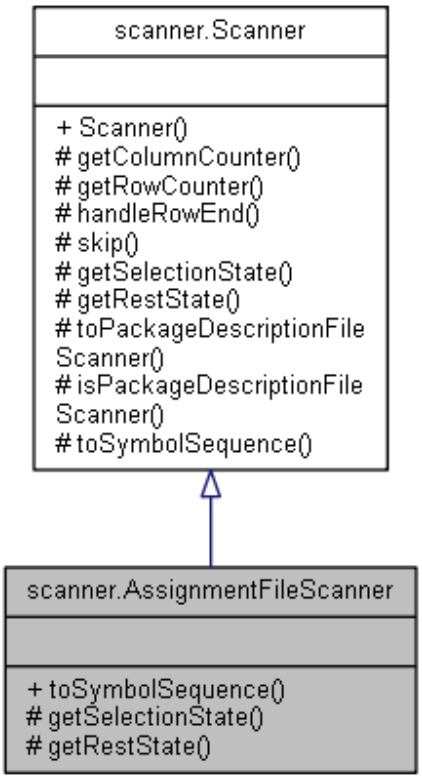
myScanner	is the used Scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the Begin Row of the Symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

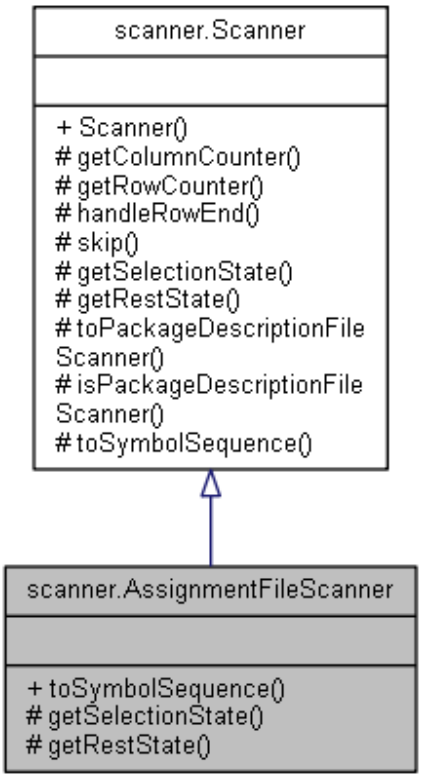
- workspace/ipfli4l13/src/scanner/AssignmentFileRestState.java

scanner.AssignmentFileScanner Klassenreferenz

scanner.AssignmentFileScannerKlassendiagramm für scanner.AssignmentFileScanner:



Zusammengehörigkeiten von scanner.AssignmentFileScanner:



Öffentliche, statische Methoden

- static void toSymbolSequence (final InputStream input, final String dataPath, final Buffer< AbstractSymbol > buffer) throws ScannerException, InterruptedException, IOException

Geschützte Methoden

- AbstractState getSelectionState ()
- AbstractState **getRestState** (final Integer beginningRow, final Integer beginningColumn)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the AssignmentFileScanner.

Autor:

HFW410

Dokumentation der Elementfunktionen

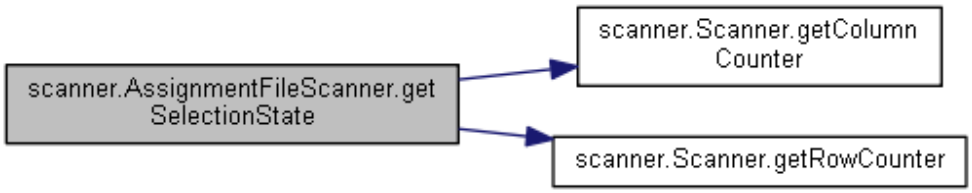
AbstractState scanner.AssignmentFileScanner.getSelectionState ()[protected], [virtual]

Rückgabe:

the SelectionState of the special scanner.

Implementiert scanner.Scanner.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



static void scanner.AssignmentFileScanner.toSymbolSequence (final InputStream *input*, final String *dataPath*, final Buffer< AbstractSymbol > *buffer*) throws ScannerException, InterruptedException, IOException `[static]`

creates a scanner and starts it.

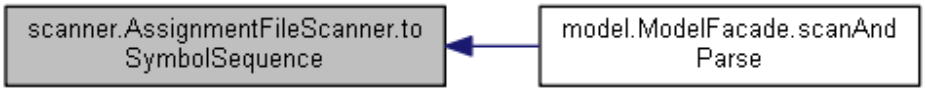
Parameter:

input	is the input stream.
dataPath	is the path of the file.
buffer	if the buffer for the symbols.

Ausnahmebehandlung:

IOException	from the Input Stream.
InterruptedException	from the buffer.
ScannerException	if there is any problem during the scan.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

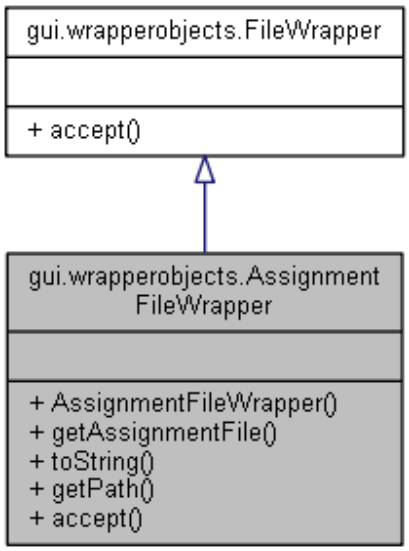


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

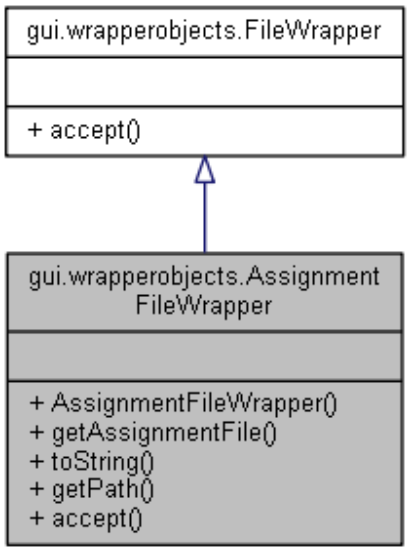
- workspace/ipfli4l13/src/scanner/AssignmentFileScanner.java

gui.wrapperobjects.AssignmentFileWrapper Klassenreferenz

gui.wrapperobjects.AssignmentFileWrapperKlassendiagramm für gui.wrapperobjects.AssignmentFileWrapper:



Zusammengehörigkeiten von gui.wrapperobjects.AssignmentFileWrapper:



Öffentliche Methoden

- AssignmentFileWrapper (final AssignmentFile assignmentFile, final String path)
- AssignmentFile getAssignmentFile ()
- String **toString** ()
- String getPath ()

- void accept (final FileWrapperVisitor vis)

Ausführliche Beschreibung

A Wrapperclass for AssignmentFile.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.wrapperobjects.AssignmentFileWrapper.AssignmentFileWrapper (final AssignmentFile *assignmentFile*, final String *path*)

Constructor for AssignmentFileWrapper.

Parameter:

path	The path to set.
assignmentFile	The assignmentfile to wrap.

Dokumentation der Elementfunktionen

void gui.wrapperobjects.AssignmentFileWrapper.accept (final FileWrapperVisitor *vis*)

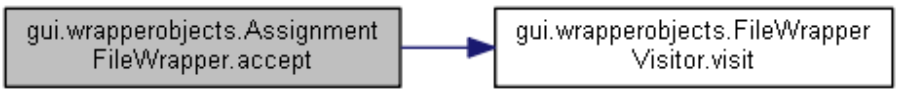
accept-method for visitor pattern.

Parameter:

vis	the visitor.

Implementiert gui.wrapperobjects.FileWrapper.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



AssignmentFile gui.wrapperobjects.AssignmentFileWrapper.getAssignmentFile ()

Rückgabe:

the typeFile

String gui.wrapperobjects.AssignmentFileWrapper.getPath ()

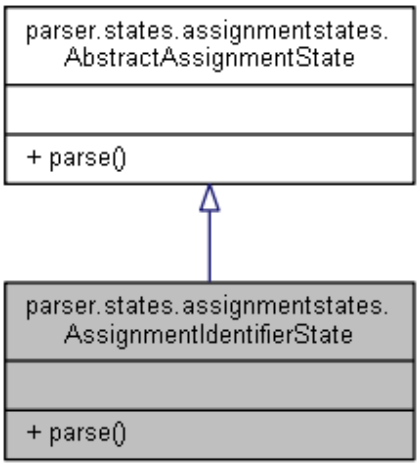
Rückgabe:
The path.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

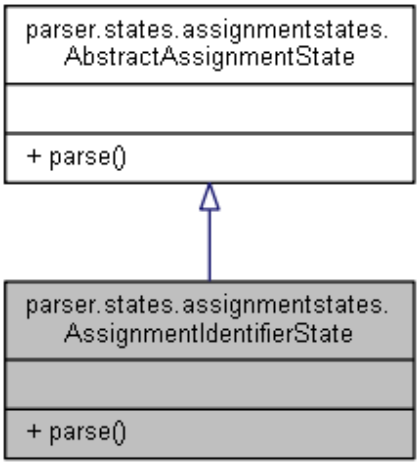
- workspace/ipfli4l13/src/gui/wrapperobjects/AssignmentFileWrapper.java

parser.states.assignmentstates.AssignmentIdentifierState Klassenreferenz

parser.states.assignmentstates.AssignmentIdentifierStateKlassendiagramm für parser.states.assignmentstates.AssignmentIdentifierState:



Zusammengehörigkeiten von parser.states.assignmentstates.AssignmentIdentifierState:



Öffentliche Methoden

- void parse (final TempVariableAssignment tVA, final Buffer< AbstractSymbol > buffer, final AssignmentFileParser aFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.assignmentstates.AssignmentIdentifierState.parse (final TempVariableAssignment *tVA*, final Buffer< AbstractSymbol > *buffer*, final AssignmentFileParser *aFP*) throws InterruptedException, ParserException

Parameter:

tVA	TempVariableAssignment
buffer	Buffer<Symbol>
aFP	AssignmentFileParser

Ausnahmebehandlung:

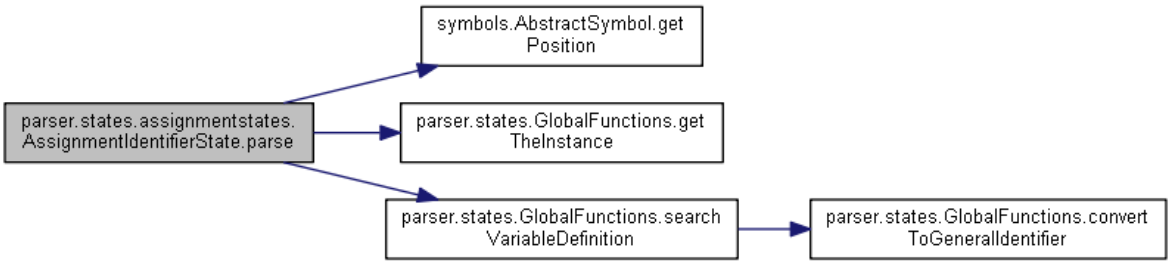
InterruptedException	InterruptedException

Rückgabe:

Ausnahmebehandlung:

ParserException	ParserException

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

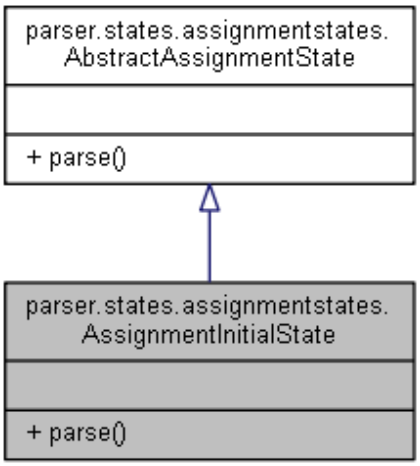


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

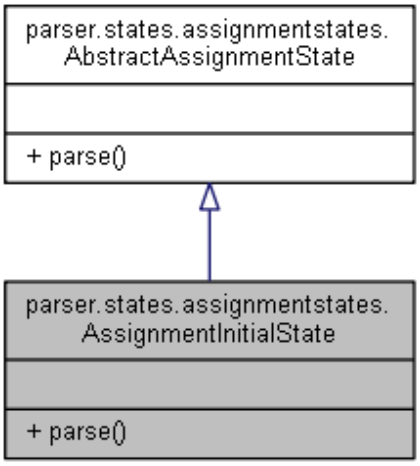
- workspace/ipfli4l13/src/parser/states/assignmentstates/AssignmentIdentifierState.java

parser.states.assignmentstates.AssignmentInitialState Klassenreferenz

parser.states.assignmentstates.AssignmentInitialStateKlassendiagramm für parser.states.assignmentstates.AssignmentInitialState:



Zusammengehörigkeiten von parser.states.assignmentstates.AssignmentInitialState:



Öffentliche Methoden

- void parse (final TempVariableAssignment tVA, final Buffer< AbstractSymbol > buffer, final AssignmentFileParser aFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.assignmentstates.AssignmentInitialState.parse (final TempVariableAssignment *tVA*, final Buffer< AbstractSymbol > *buffer*, final AssignmentFileParser *aFP*) throws InterruptedException, ParserException

Parameter:

tVA	TempVariableAssignment
buffer	Buffer<Symbol>
aFP	AssignmentFileParser

Ausnahmebehandlung:

InterruptedException	InterruptedException

Rückgabe:

Ausnahmebehandlung:

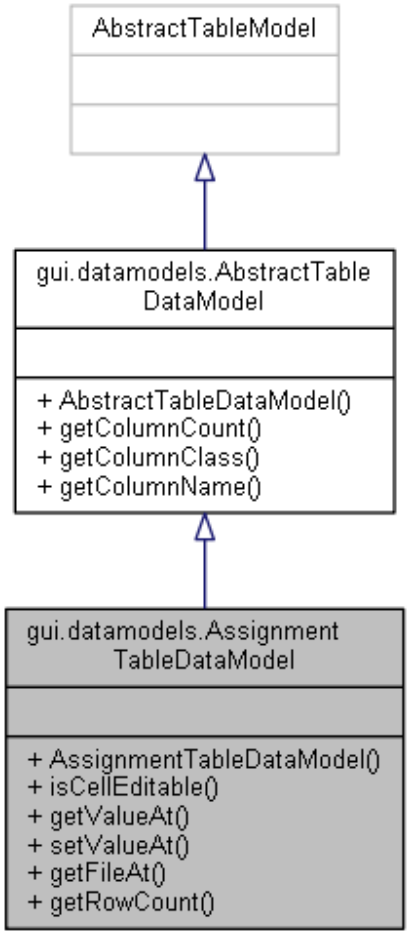
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

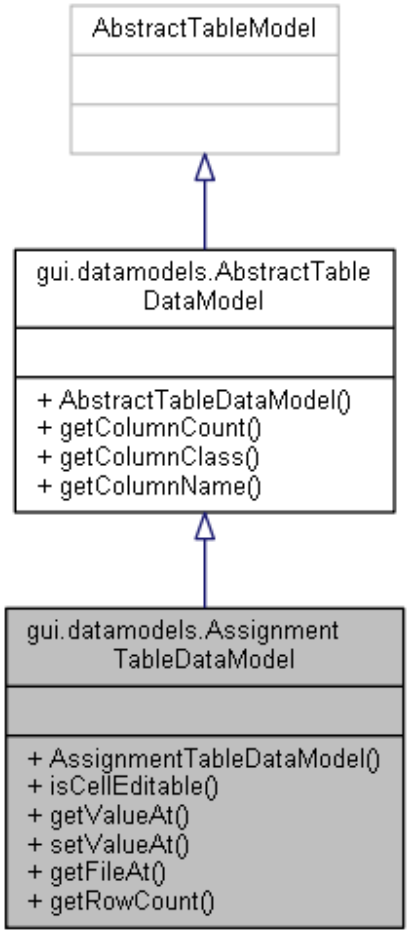
- workspace/ipfli4l13/src/parser/states/assignmentstates/AssignmentInitialState.java

gui.datamodels.AssignmentTableDataModel Klassenreferenz

gui.datamodels.AssignmentTableDataModelKlassendiagramm für gui.datamodels.AssignmentTableDataModel:



Zusammengehörigkeiten von gui.datamodels.AssignmentTableDataModel:



Öffentliche Methoden

- AssignmentTableDataModel (final List< VariableAssignment > data)
- boolean **isCellEditable** (final int r, final int c)
- Object **getValueAt** (final int r, final int c)
- void setValueAt (final Object value, final int row, final int col)
- VariableAssignment getFileAt (final int r)
- int **getRowCount** ()

Ausführliche Beschreibung

tableModel for assignmentfiles.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.datamodels.AssignmentTableDataModel.AssignmentTableDataModel (final List< VariableAssignment > *data*)

Constructor, wraps the given data into a list of wraped data and sorts the list by the positions of entries.

Parameter:

data	The data to set.

Dokumentation der Elementfunktionen

VariableAssignment gui.datamodels.AssignmentTableDataModel.getFileAt (final int *r*)

Returns the VariableAssignment of the row r.

Parameter:

r	The row r.

Rückgabe:

the variable assignment.

void gui.datamodels.AssignmentTableDataModel.setValueAt (final Object *value*, final int *row*, final int *col*)

Parameter:

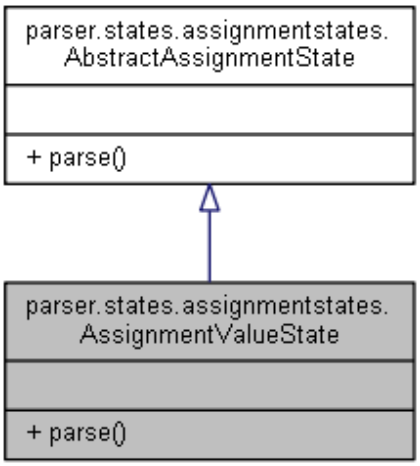
value	The new value to set.
row	The new row to set.
col	The new col to set.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

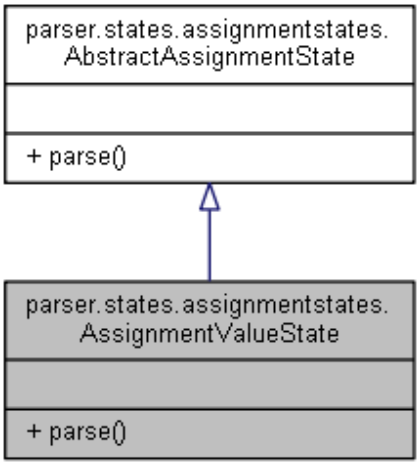
- workspace/ipfli4l13/src/gui/datamodels/AssignmentTableDataModel.java

parser.states.assignmentstates.AssignmentValueState Klassenreferenz

parser.states.assignmentstates.AssignmentValueStateKlassendiagramm für parser.states.assignmentstates.AssignmentValueState:



Zusammengehörigkeiten von parser.states.assignmentstates.AssignmentValueState:



Öffentliche Methoden

- void parse (final TempVariableAssignment tVA, final Buffer< AbstractSymbol > buffer, final AssignmentFileParser aFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.assignmentstates.AssignmentValueState.parse (final TempVariableAssignment *tVA*, final Buffer< AbstractSymbol > *buffer*, final AssignmentFileParser *aFP*) throws InterruptedException, ParserException

Parameter:

tVA	TempVariableAssignment
buffer	Buffer<Symbol>
aFP	AssignmentFileParser

Ausnahmebehandlung:

InterruptedException	InterruptedException

Rückgabe:

Ausnahmebehandlung:

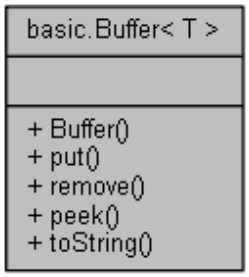
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/states/assignmentstates/AssignmentValueState.java

basic.Buffer< T > Klassenreferenz

basic.Buffer< T >Zusammengehörigkeiten von basic.Buffer< T >:



Öffentliche Methoden

- Buffer (final Integer capacity)
- void put (final T object) throws InterruptedException
- T remove () throws InterruptedException
- T peek () throws InterruptedException
- String **toString** ()

Ausführliche Beschreibung

This class represents a Buffer and will be used to send the data from scanner to parser.

Autor:

HFW410

Parameter:

<T>	

Beschreibung der Konstruktoren und Destruktoren

basic.Buffer< T >.Buffer (final Integer *capacity*)

This is the constructor. It initialize a new buffer with the capacity

Parameter:

capacity	No side-effects.

Dokumentation der Elementfunktionen

T basic.Buffer< T >.peek () throws InterruptedException

Rückgabe:

the first element

Ausnahmebehandlung:

InterruptedException	if mutex.down did not work

void basic.Buffer< T >.put (final T *object*) throws InterruptedException

This method put a new object

Parameter:

object	into the buffer.

Ausnahmebehandlung:

InterruptedException,if	mutex.down does not work

T basic.Buffer< T >.remove () throws InterruptedException

removes the first element of the buffer.

Rückgabe:

the removed data

Ausnahmebehandlung:

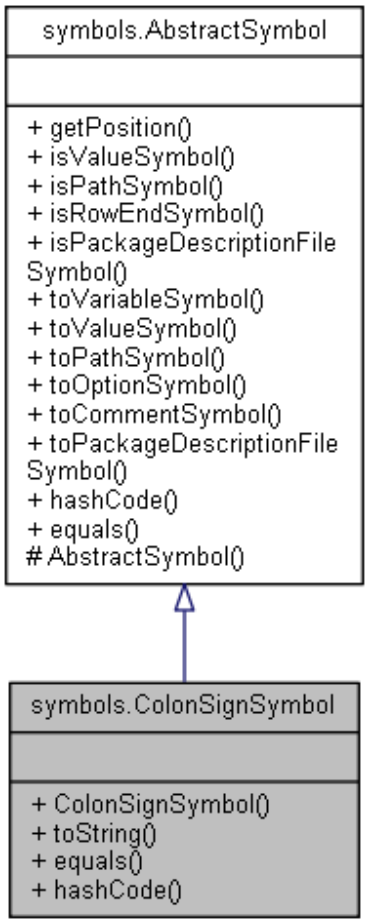
InterruptedException	if mutex.down() did not work

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

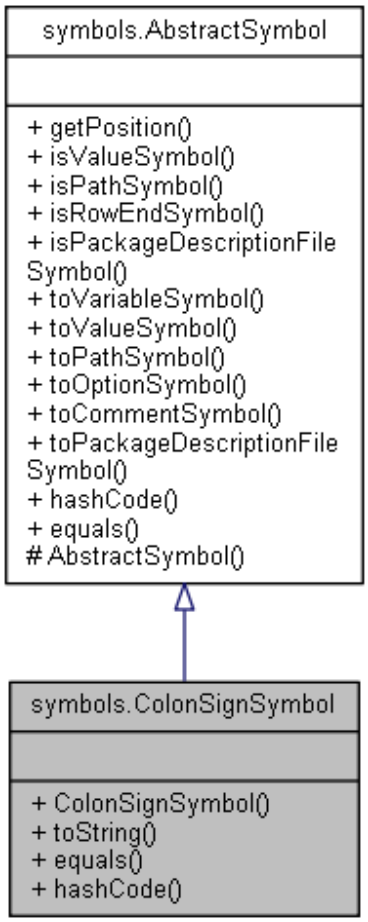
- workspace/ipfli4l13/src/basic/Buffer.java

symbols.ColonSignSymbol Klassenreferenz

symbols.ColonSignSymbolKlassendiagramm für symbols.ColonSignSymbol:



Zusammengehörigkeiten von symbols.ColonSignSymbol:



Öffentliche Methoden

- ColonSignSymbol (final Position position)
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the colonsign. It implements the Interface Symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.ColonSignSymbol.ColonSignSymbol (final Position *position*)

This constructor instantiate the colonsign.

Parameter:

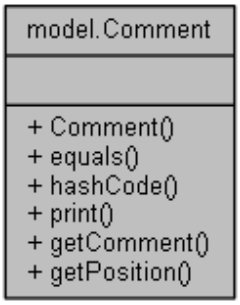
position	is the position of the data in the stream.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/symbols/ColonSignSymbol.java

model.Comment Klassenreferenz

model.CommentZusammengehörigkeiten von model.Comment:



Öffentliche Methoden

- Comment (final String comment, final Position position)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- String print ()
- String getComment ()
- Position getPosition ()

Ausführliche Beschreibung

Autor:

Muri

Beschreibung der Konstruktoren und Destruktoren

model.Comment.Comment (final String *comment*, final Position *position*)

A Comment in a fli4l configuration File. A HashMap of these are stored in the package Manager.

Parameter:

comment	Comment
position	Position

Dokumentation der Elementfunktionen

String model.Comment.getComment ()

Rückgabe:

the comment. no side effects.

Position model.Comment.getPosition ()

Rückgabe:
the position. No side effects.

String model.Comment.print ()

Returns the line to print.

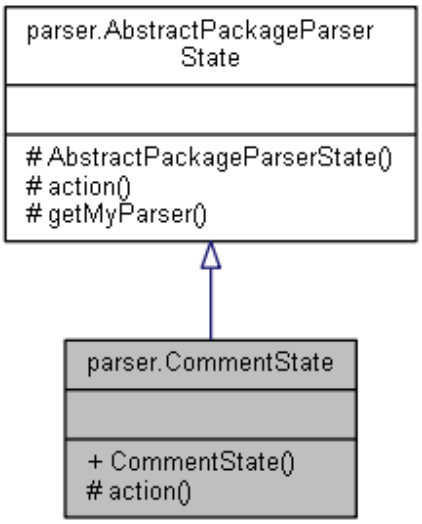
Rückgabe:
Returns a string value, representing a line to print.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

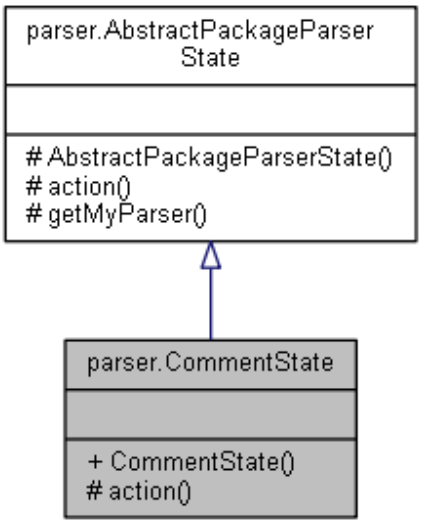
- workspace/ipfli4l13/src/model/Comment.java

parser.CommentState Klassenreferenz

parser.CommentStateKlassendiagramm für parser.CommentState:



Zusammengehörigkeiten von parser.CommentState:



Öffentliche Methoden

- CommentState (final PackageDescriptionFileParser parser)

Geschützte Methoden

- AbstractPackageParserState **action** (final PackageDescriptionFileSymbol symbol, final Map< String, AbstractVariableDefinition > variableDef) throws InterruptedException, ParserException

Ausführliche Beschreibung

This class represents the CommentState of the PackageDescriptionfileParser.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

parser.CommentState.CommentState (final PackageDescriptionFileParser *parser*)

constructor.

Parameter:

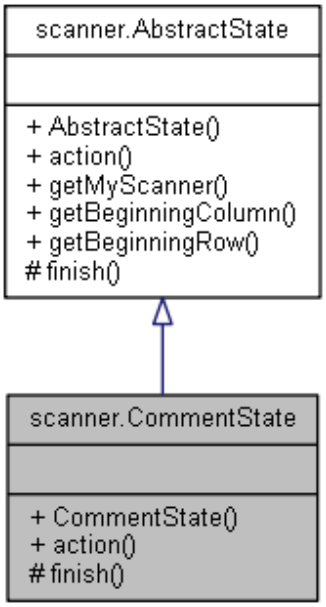
parser	the PackageDescriptionfileParser.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

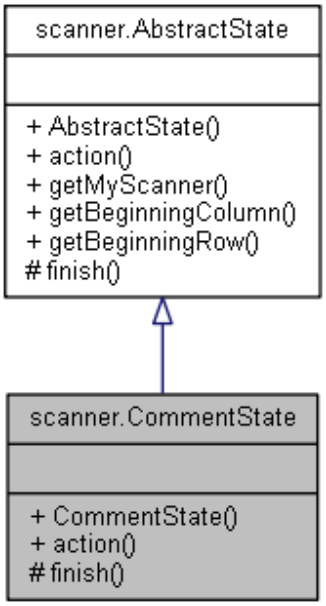
- workspace/ipfli4l13/src/parser/CommentState.java

scanner.CommentState Klassenreferenz

scanner.CommentStateKlassendiagramm für scanner.CommentState:



Zusammengehörigkeiten von scanner.CommentState:



Öffentliche Methoden

- CommentState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)

- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

This class represents the commentState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.CommentState.CommentState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

This is the constructor for a new CommentState.

Parameter:

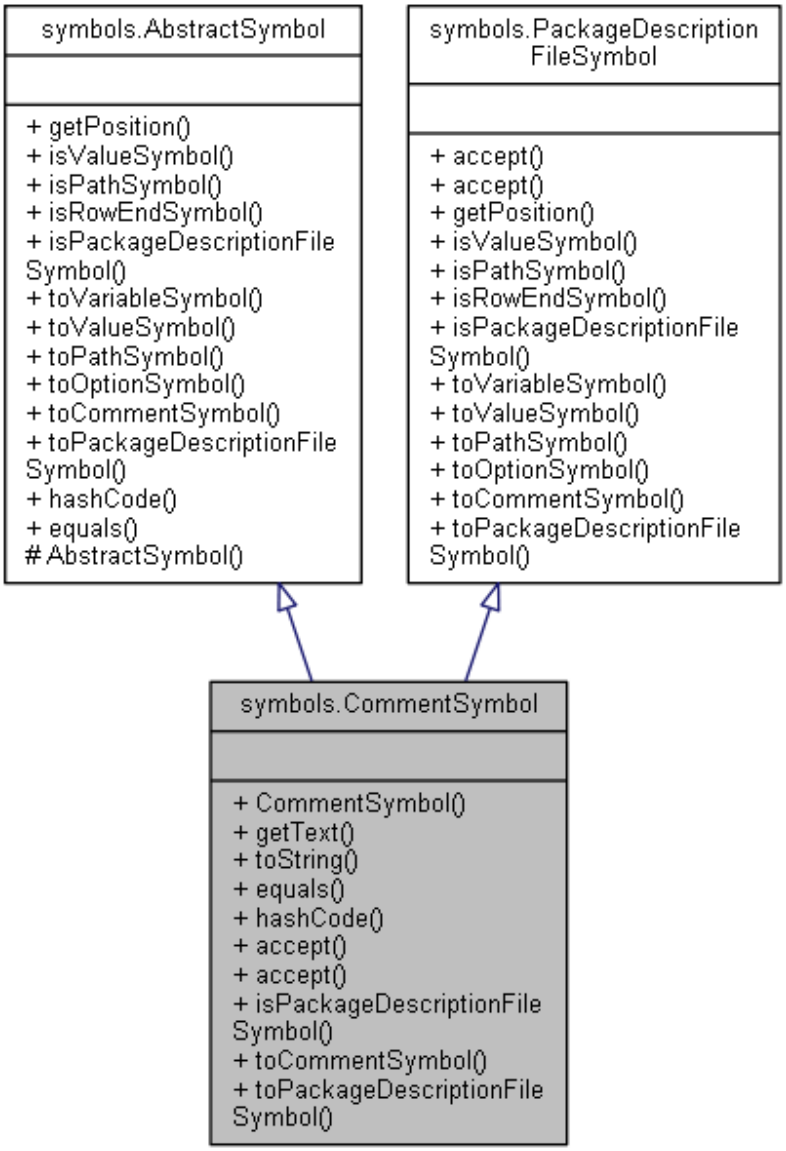
myScanner	is the used Scanner.
beginningColumn	is the Begin Column of the symbol
beginningRow	is the begin row of the symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

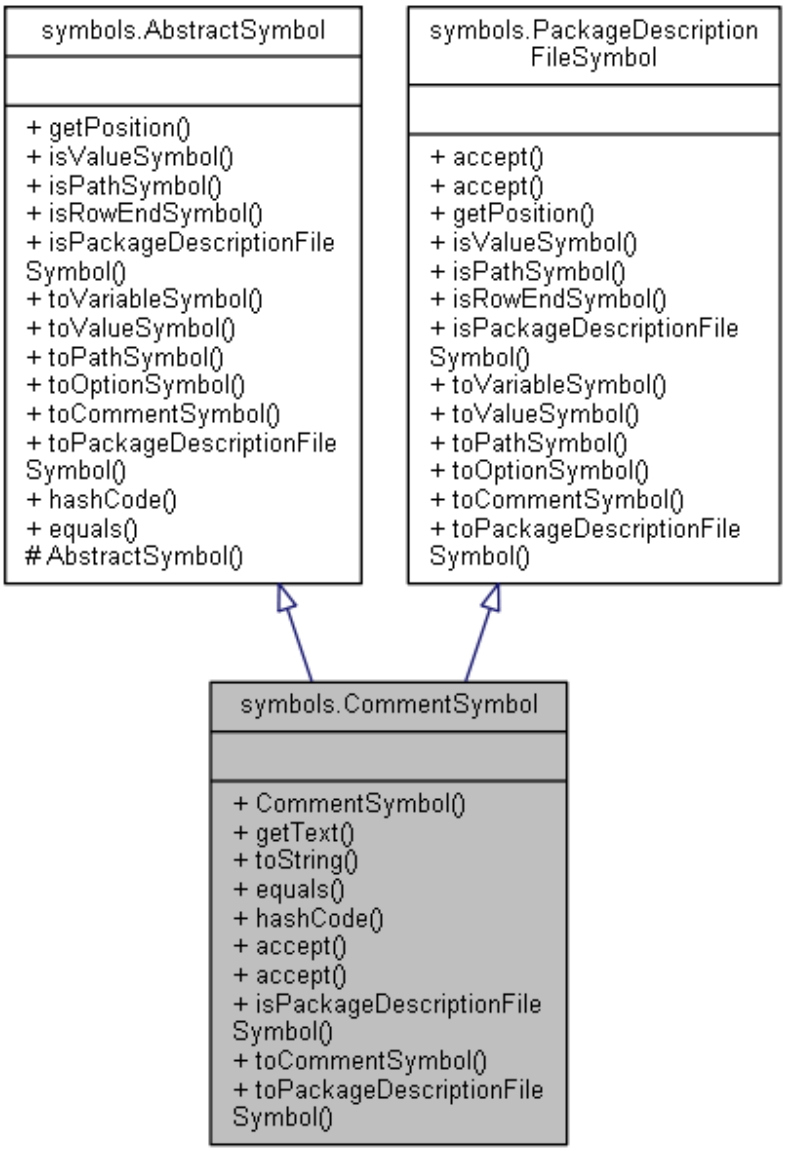
- workspace/ipfli4l13/src/scanner/CommentState.java

symbols.CommentSymbol Klassenreferenz

symbols.CommentSymbolKlassendiagramm für symbols.CommentSymbol:



Zusammengehörigkeiten von symbols.CommentSymbol:



Öffentliche Methoden

- CommentSymbol (final String text, final Position position)
- String getText ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitor visitor, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitorWithTempRule visitor, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- Boolean isPackageDescriptionFileSymbol ()
- CommentSymbol toCommentSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the Symbol of a Comment.

Beschreibung der Konstruktoren und Destruktoren

symbols.CommentSymbol.CommentSymbol (final String *text*, final Position *position*)

This constructor instantiate a new CommentSymbol with

Parameter:

text.	
text	is the text of the comment.
position	is the position of the data in the stream.

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.CommentSymbol.accept (final PackageDescriptionSymbolVisitor *visitor*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

AbstractPackageParserState symbols.CommentSymbol.accept (final PackageDescriptionSymbolVisitorWithTempRule *visitor*, final

PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackage-Rule*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.
tempPackageRule	is a temporary package rule.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

String symbols.CommentSymbol.getText ()

This Operation

Rückgabe:

the commenttext.

Boolean symbols.CommentSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

CommentSymbol symbols.CommentSymbol.toCommentSymbol ()

Rückgabe:

an error, if the symbol is not a comment symbol

Implementiert symbols.PackageDescriptionFileSymbol.

PackageDescriptionFileSymbol symbols.CommentSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:

a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.

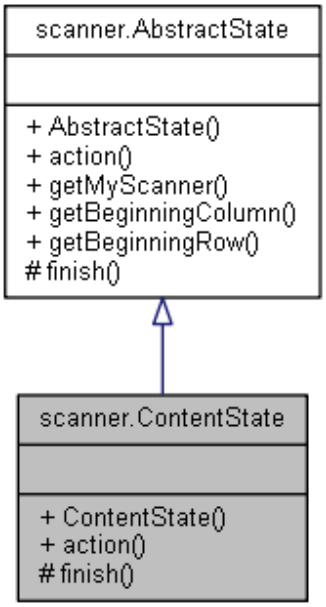
Implementiert symbols.PackageDescriptionFileSymbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

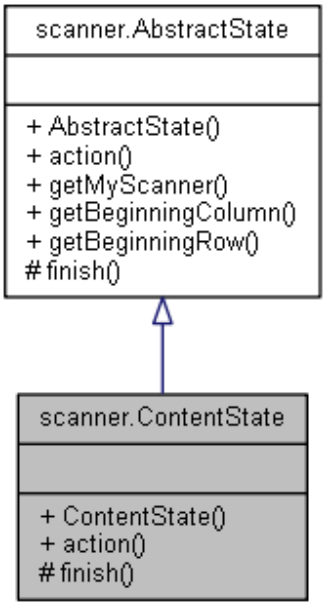
- workspace/ipfli4l13/src/symbols/CommentSymbol.java

scanner.ContentState Klassenreferenz

scanner.ContentStateKlassendiagramm für scanner.ContentState:



Zusammengehörigkeiten von scanner.ContentState:



Öffentliche Methoden

- ContentState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)

- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

State for the Content in the PackageDescriptionFile.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

scanner.ContentState.ContentState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

creates the Pathstate for the PackageDesriptionFileScanner.

Parameter:

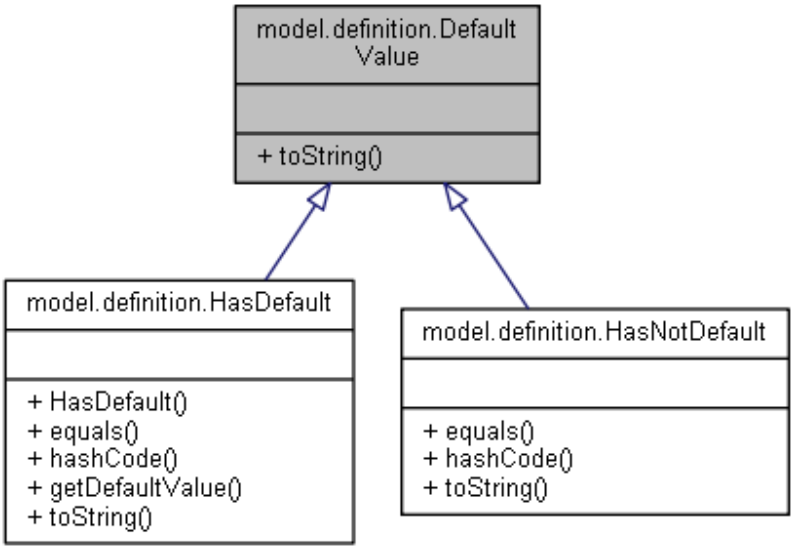
myScanner	is the scanner, which owns the state.
beginningColumn	is the first column of the symbol.
beginningRow	is the first row of the symbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

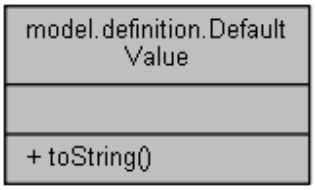
- workspace/ipfli4l13/src/scanner/ContentState.java

model.definition.DefaultValue Schnittstellenreferenz

model.definition.DefaultValueKlassendiagramm für model.definition.DefaultValue:



Zusammengehörigkeiten von model.definition.DefaultValue:



Öffentliche Methoden

- String **toString** ()

Ausführliche Beschreibung

Autor:

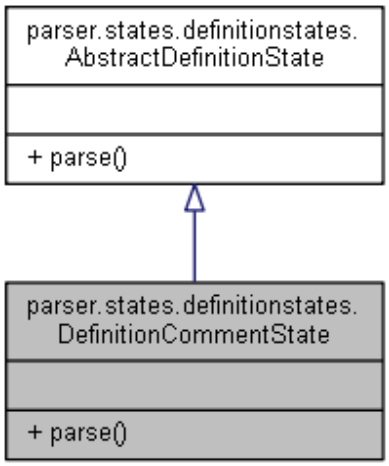
HFW410RA - Philipp Rammos

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

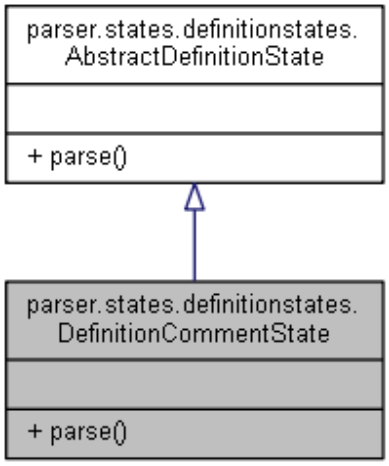
- workspace/ipfli4l13/src/model/definition/DefaultValue.java

parser.states.definitionstates.DefinitionCommentState Klassenreferenz

parser.states.definitionstates.DefinitionCommentStateKlassendiagramm für parser.states.definitionstates.DefinitionCommentState:



Zusammengehörigkeiten von parser.states.definitionstates.DefinitionCommentState:



Öffentliche Methoden

- void parse (final TempVariableDefinition tVD, final Buffer< AbstractSymbol > buffer, final DefinitionFileParser dFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.definitionstates.DefinitionCommentState.parse (final TempVariableDefinition *tVD*, final Buffer< AbstractSymbol > *buffer*, final DefinitionFileParser *dFP*) throws InterruptedException, ParserException

Parameter:

tVD	TempVariableDefinition
dFP	DefinitionFileParser
buffer	Buffer<Symbol>

Rückgabe:

Ausnahmebehandlung:

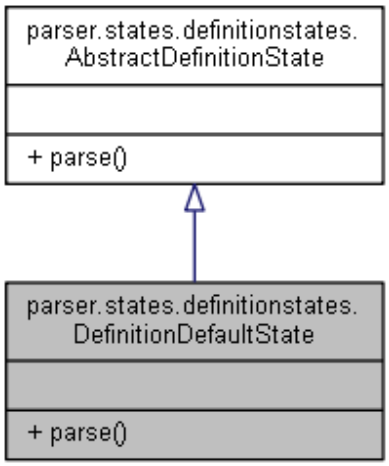
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

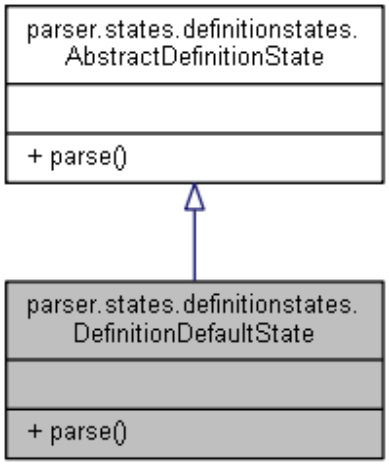
- workspace/ipfli4l13/src/parser/states/definitionstates/DefinitionCommentState.java

parser.states.definitionstates.DefinitionDefaultState Klassenreferenz

parser.states.definitionstates.DefinitionDefaultStateKlassendiagramm für parser.states.definitionstates.DefinitionDefaultState:



Zusammengehörigkeiten von parser.states.definitionstates.DefinitionDefaultState:



Öffentliche Methoden

- void parse (final TempVariableDefinition tVD, final Buffer< AbstractSymbol > buffer, final DefinitionFileParser dFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.definitionstates.DefinitionDefaultState.parse (final TempVariableDefinition *tVD*, final Buffer< AbstractSymbol > *buffer*, final DefinitionFileParser *dFP*) throws InterruptedException, ParserException

Parameter:

tVD	TempVariableDefinition
dFP	DefinitionFileParser
buffer	Buffer<Symbol>

Rückgabe:

Ausnahmebehandlung:

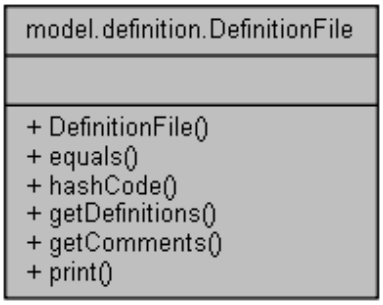
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/states/definitionstates/DefinitionDefaultState.java

model.definition.DefinitionFile Klassenreferenz

model.definition.DefinitionFileZusammengehörigkeiten von model.definition.DefinitionFile:



Öffentliche Methoden

- DefinitionFile (final Map< String, AbstractVariableDefinition > definitions2, final List< Comment > comments)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- java.util.Map< String,
- AbstractVariableDefinition > getDefinitions ()
- java.util.List< Comment > getComments ()
- String print (final String pathToPrint) throws printer.PrinterException

---

Ausführliche Beschreibung

DefinitionFileFile represents the "\check\<package>.txt" files.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.definition.DefinitionFile.DefinitionFile (final Map< String, AbstractVariableDefinition > *definitions2*, final List< Comment > *comments*)

Constructor, just sets the fields.

Parameter:

definitions2	The definitions to set.
comments	The comments to set.

Dokumentation der Elementfunktionen

java.util.List<Comment> model.definition.DefinitionFile.getComments ()

Rückgabe:

the comments

java.util.Map<String, AbstractVariableDefinition> model.definition.DefinitionFile.getDefinitions ()

Returns the field definitions. No side effects.

Rückgabe:

this.definitions

String model.definition.DefinitionFile.print (final String *pathToPrint*) throws printer.PrinterException

Prints a DefinitionFile containing the VariableDefinitions and Comments.

Parameter:

pathToPrint	The path where to print the file.

Ausnahmebehandlung:

printer.PrinterException	If a IOException is thrown.
printer.PrinterException	

Rückgabe:

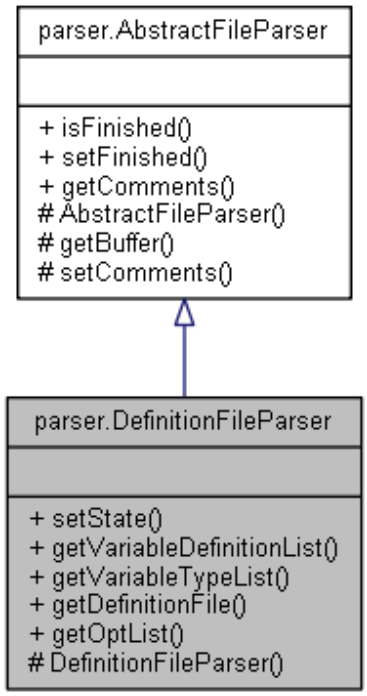
Returns a string representing the printed lines (for testing).

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

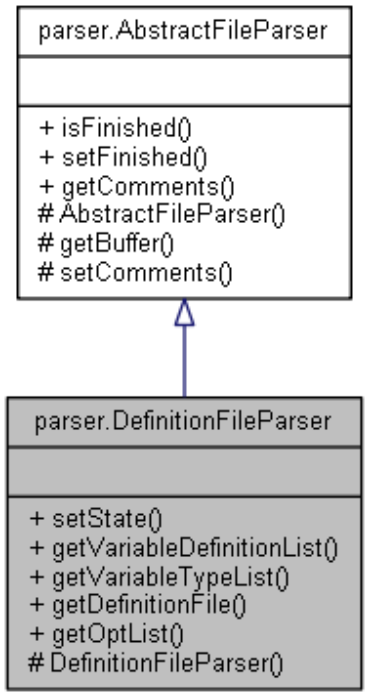
- workspace/ipfli4l13/src/model/definition/DefinitionFile.java

parser.DefinitionFileParser Klassenreferenz

parser.DefinitionFileParserKlassendiagramm für parser.DefinitionFileParser:



Zusammengehörigkeiten von parser.DefinitionFileParser:



Öffentliche Methoden

- void setState (final AbstractDefinitionState state)
- Map< String,
- AbstractVariableDefinition > getVariableDefinitionList ()
- Map< String, NamedVariableType > getVariableTypeList ()
- DefinitionFile getDefinitionFile ()
- Map< String, Opt > getOptList ()

Geschützte Methoden

- DefinitionFileParser (final Buffer< AbstractSymbol > buffer, final Map< String, NamedVariableType > map) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:

hfw410wi - Mark Wittig

Beschreibung der Konstruktoren und Destruktoren

parser.DefinitionFileParser.DefinitionFileParser (final Buffer< AbstractSymbol > *buffer*, final Map< String, NamedVariableType > *map*) throws InterruptedException, ParserException `[protected]`

The DefinitionFileParser will automatically generate a DefinitionFile after instantiation which can be obtained via getDefinitionFile().

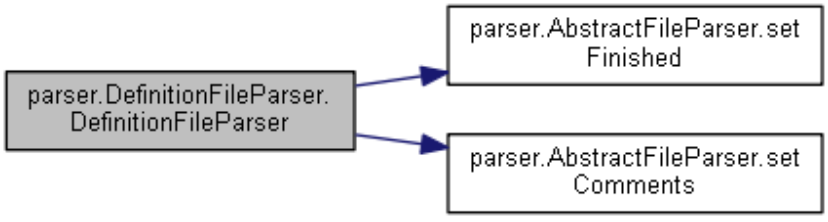
Parameter:

buffer	Buffer<Symbol>
map	VariableTypeList

Ausnahmebehandlung:

ParserException	ParserExcpetion
InterruptedException	InterruptedException

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Dokumentation der Elementfunktionen

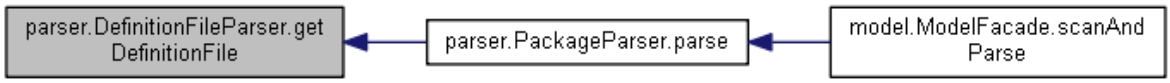
DefinitionFile parser.DefinitionFileParser.getDefinitionFile ()

Returns the desired DefinitionFile which is automatically created right after instantiation.

Rückgabe:

the definitionFile

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

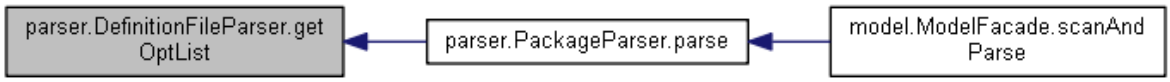


Map<String, Opt> parser.DefinitionFileParser.getOptList ()

Rückgabe:

the optList

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



Map<String, AbstractVariableDefinition> parser.DefinitionFileParser.getVariableDefinitionList ()

Rückgabe:

the variableDefinitionList

Map<String, NamedVariableType> parser.DefinitionFileParser.getVariableTypeList ()

Rückgabe:

the variableTypeList

void parser.DefinitionFileParser.setState (final AbstractDefinitionState *state*)

Parameter:

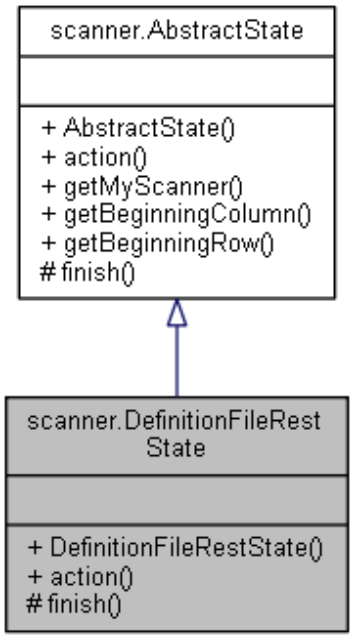
state	the state to set

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

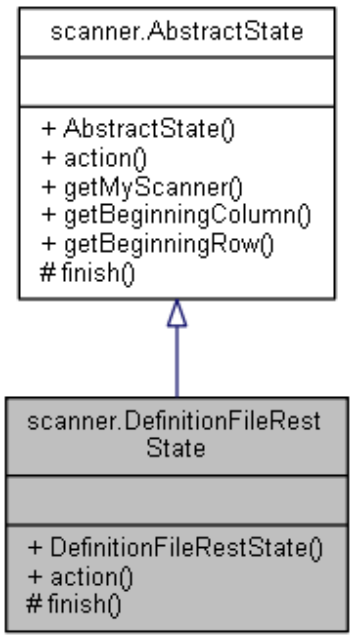
- workspace/ipfli4l13/src/parser/DefinitionFileParser.java

scanner.DefinitionFileRestState Klassenreferenz

scanner.DefinitionFileRestStateKlassendiagramm für scanner.DefinitionFileRestState:



Zusammengehörigkeiten von scanner.DefinitionFileRestState:



Öffentliche Methoden

- DefinitionFileRestState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)
- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath)

Ausführliche Beschreibung

This class represents the RestState for the DefinitionFileScanner.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.DefinitionFileRestState.DefinitionFileRestState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

initialize the DefinitionFileRestState.

Parameter:

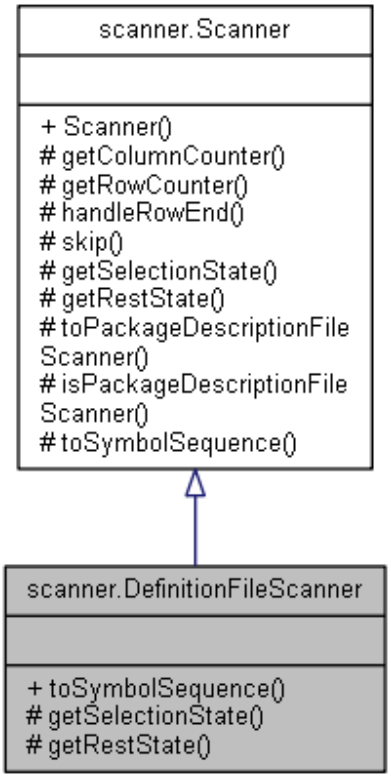
myScanner	used Scanner
beginningColumn	the column where is the first character
beginningRow	is the row where is the first character

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

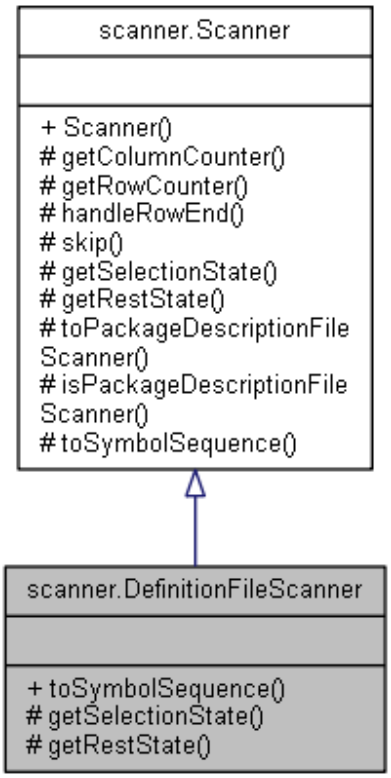
- workspace/ipfli4l13/src/scanner/DefinitionFileRestState.java

scanner.DefinitionFileScanner Klassenreferenz

scanner.DefinitionFileScannerKlassendiagramm für scanner.DefinitionFileScanner:



Zusammengehörigkeiten von scanner.DefinitionFileScanner:



Öffentliche, statische Methoden

- static void toSymbolSequence (final InputStream input, final String dataPath, final Buffer< AbstractSymbol > buffer) throws ScannerException, InterruptedException, IOException

Geschützte Methoden

- AbstractState getSelectionState ()
- AbstractState **getRestState** (final Integer beginningRow, final Integer beginningColumn)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the DefinitionFileScanner.

Autor:

HFW410

Dokumentation der Elementfunktionen

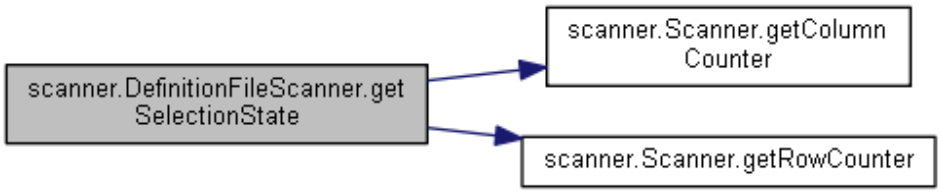
AbstractState scanner.DefinitionFileScanner.getSelectionState ()[protected], [virtual]

Rückgabe:

the SelectionState of the special scanner.

Implementiert scanner.Scanner.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



static void scanner.DefinitionFileScanner.toSymbolSequence (final InputStream *input*, final String *dataPath*, final Buffer< AbstractSymbol > *buffer*) throws ScannerException, InterruptedException, IOException `[static]`

creates a scanner and starts him.

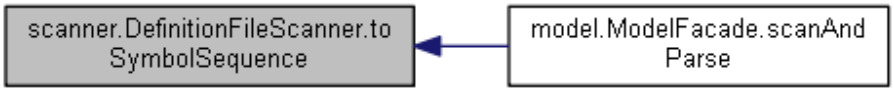
Parameter:

input	is the input stream.
dataPath	is the path of the file.
buffer	if the buffer for the symbols.

Ausnahmebehandlung:

IOException	from the Input Stream.
InterruptedException	from the buffer.
ScannerException	if there is any problem during the scan.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

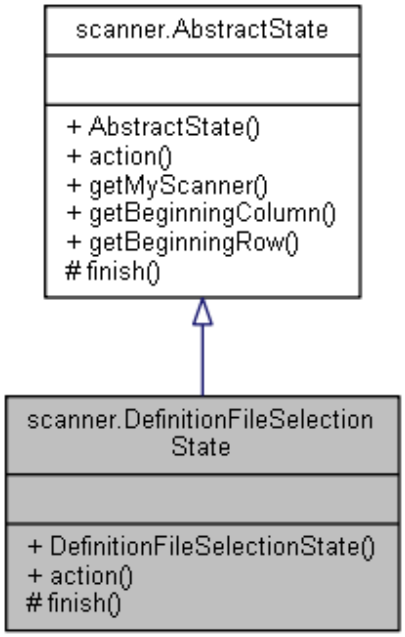


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

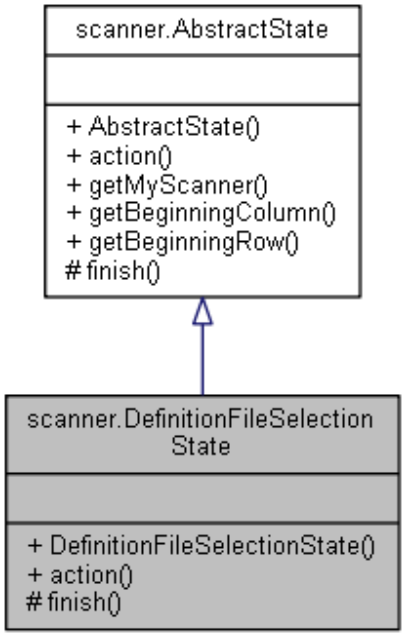
- workspace/ipfli4l13/src/scanner/DefinitionFileScanner.java

scanner.DefinitionFileSelectionState Klassenreferenz

scanner.DefinitionFileSelectionStateKlassendiagramm für scanner.DefinitionFileSelectionState:



Zusammengehörigkeiten von scanner.DefinitionFileSelectionState:



Öffentliche Methoden

- DefinitionFileSelectionState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)
- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath)

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath)

Ausführliche Beschreibung

This class represents the DefinitionFileSelectionState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.DefinitionFileSelectionState.DefinitionFileSelectionState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

This constructor initialize the new DefinitionFileSelectionState.

Parameter:

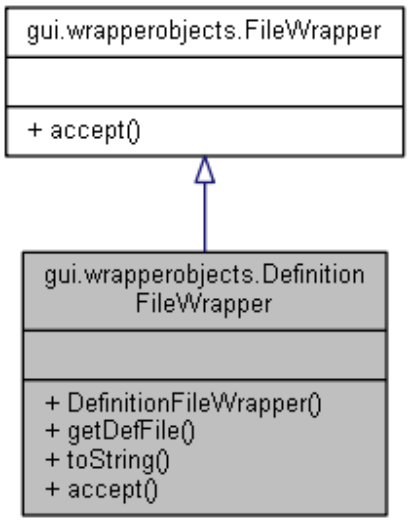
myScanner	is the used scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the beginning row of the symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

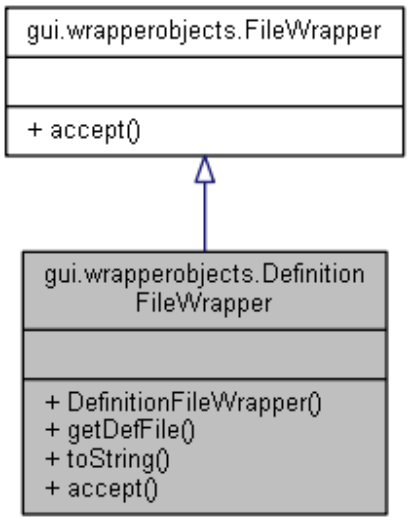
- workspace/ipfli4l13/src/scanner/DefinitionFileSelectionState.java

gui.wrapperobjects.DefinitionFileWrapper Klassenreferenz

gui.wrapperobjects.DefinitionFileWrapperKlassendiagramm für gui.wrapperobjects.DefinitionFileWrapper:



Zusammengehörigkeiten von gui.wrapperobjects.DefinitionFileWrapper:



Öffentliche Methoden

- DefinitionFileWrapper (final DefinitionFile defFile)
- DefinitionFile getDefFile ()
- String **toString** ()
- void accept (final FileWrapperVisitor vis)

Ausführliche Beschreibung

A Wrapperclass for DefinitionFile.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.wrapperobjects.DefinitionFileWrapper.DefinitionFileWrapper (final DefinitionFile *defFile*)

Constructor for DefinitionFileWrapper.

Parameter:

defFile	The defFile to wrap.

Dokumentation der Elementfunktionen

void gui.wrapperobjects.DefinitionFileWrapper.accept (final FileWrapperVisitor *vis*)

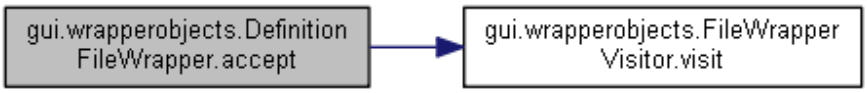
accept-method for visitor pattern.

Parameter:

vis	the visitor.

Implementiert gui.wrapperobjects.FileWrapper.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



DefinitionFile gui.wrapperobjects.DefinitionFileWrapper.getDefFile ()

Rückgabe:

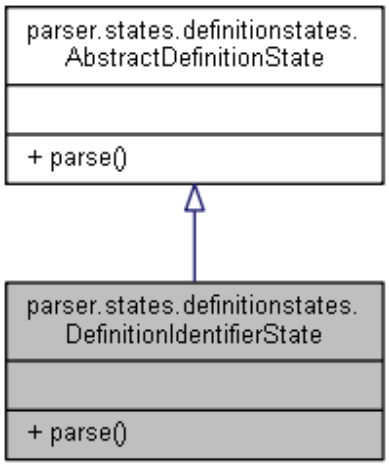
the typeFile

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

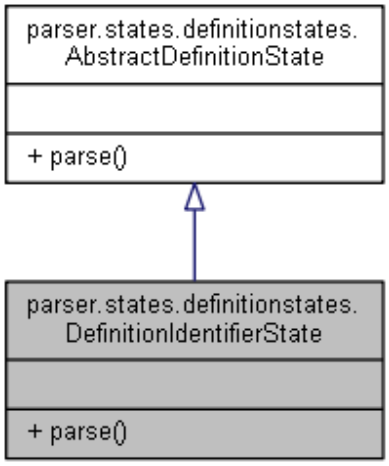
- workspace/ipfli4l13/src/gui/wrapperobjects/DefinitionFileWrapper.java

parser.states.definitionstates.DefinitionIdentifierState Klassenreferenz

parser.states.definitionstates.DefinitionIdentifierStateKlassendiagramm für parser.states.definitionstates.DefinitionIdentifierState:



Zusammengehörigkeiten von parser.states.definitionstates.DefinitionIdentifierState:



Öffentliche Methoden

- void parse (final TempVariableDefinition tVD, final Buffer< AbstractSymbol > buffer, final DefinitionFileParser dFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.definitionstates.DefinitionIdentifierState.parse (final TempVariableDefinition *tVD*, final Buffer< AbstractSymbol > *buffer*, final DefinitionFileParser *dFP*) throws InterruptedException, ParserException

Parameter:

tVD	TempVariableDefinition
dFP	DefinitionFileParser
buffer	Buffer<Symbol>

Rückgabe:

Ausnahmebehandlung:

InterruptedException	InterruptedException
ParserException	ParserException

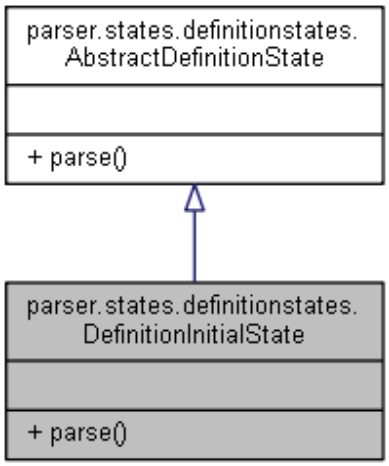
Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

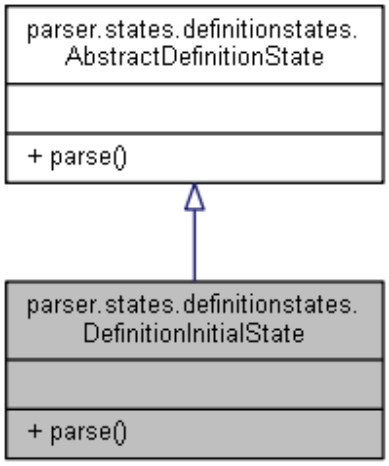
- workspace/ipfli4l13/src/parser/states/definitionstates/DefinitionIdentifierState.java

parser.states.definitionstates.DefinitionInitialState Klassenreferenz

parser.states.definitionstates.DefinitionInitialStateKlassendiagramm für parser.states.definitionstates.DefinitionInitialState:



Zusammengehörigkeiten von parser.states.definitionstates.DefinitionInitialState:



Öffentliche Methoden

- void parse (final TempVariableDefinition tVD, final Buffer< AbstractSymbol > buffer, final DefinitionFileParser dFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.definitionstates.DefinitionInitialState.parse (final TempVariableDefinition *tVD*, final Buffer< AbstractSymbol > *buffer*, final DefinitionFileParser *dFP*) throws InterruptedException, ParserException

Parameter:

tVD	TempVariableDefinition
dFP	DefinitionFileParser
buffer	Buffer<Symbol>

Rückgabe:

Ausnahmebehandlung:

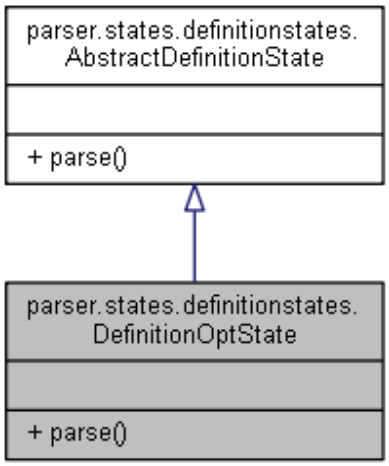
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

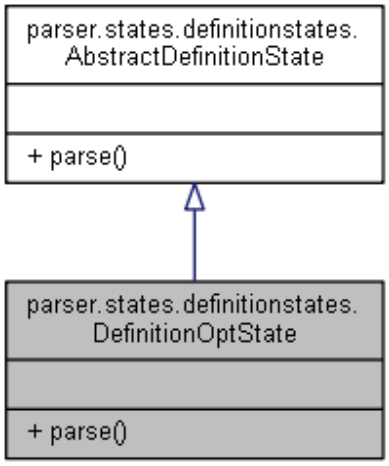
- workspace/ipfli4l13/src/parser/states/definitionstates/DefinitionInitialState.java

parser.states.definitionstates.DefinitionOptState Klassenreferenz

parser.states.definitionstates.DefinitionOptStateKlassendiagramm für parser.states.definitionstates.DefinitionOptState:



Zusammengehörigkeiten von parser.states.definitionstates.DefinitionOptState:



Öffentliche Methoden

- void parse (final TempVariableDefinition tVD, final Buffer< AbstractSymbol > buffer, final DefinitionFileParser dFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.definitionstates.DefinitionOptState.parse (final TempVariableDefinition *tVD*, final Buffer< AbstractSymbol > *buffer*, final DefinitionFileParser *dFP*) throws InterruptedException, ParserException

Parameter:

tVD	TempVariableDefinition
dFP	DefinitionFileParser
buffer	Buffer<Symbol>

Rückgabe:

Ausnahmebehandlung:

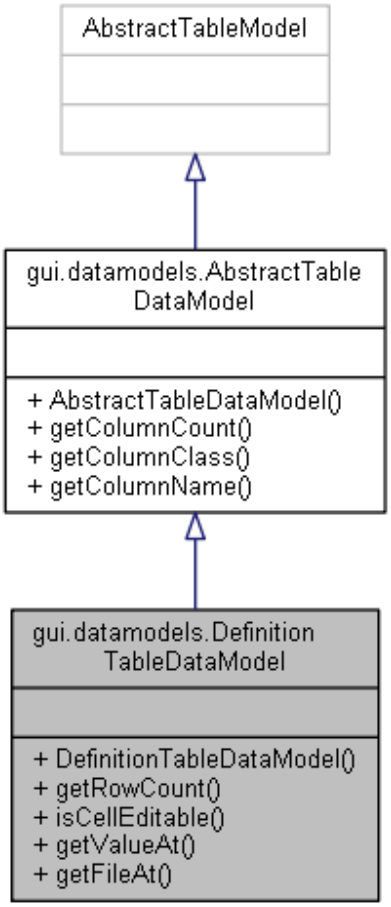
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

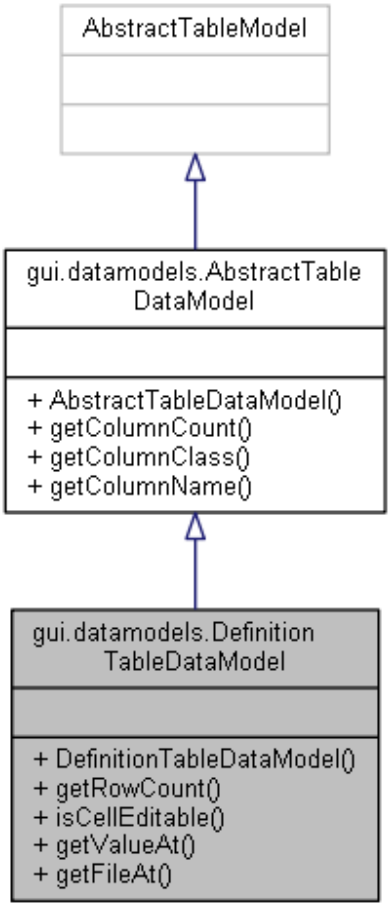
- workspace/ipfli4l13/src/parser/states/definitionstates/DefinitionOptState.java

gui.datamodels.DefinitionTableDataModel Klassenreferenz

gui.datamodels.DefinitionTableDataModelKlassendiagramm für gui.datamodels.DefinitionTableDataModel:



Zusammengehörigkeiten von gui.datamodels.DefinitionTableDataModel:



Öffentliche Methoden

- DefinitionTableDataModel (final List< AbstractVariableDefinition > data)
- int **getRowCount** ()
- boolean **isCellEditable** (final int r, final int c)
- Object **getValueAt** (final int r, final int c)
- AbstractVariableDefinition getFileAt (final int r)

Ausführliche Beschreibung

tableModel for definition table.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.datamodels.DefinitionTableDataModel.DefinitionTableDataModel (final List< AbstractVariableDefinition > *data*)

Constructor, wraps the given data into a list of wraped data and sorts the list by the positions of entries.

Parameter:

data	the data to initialize the table.

Rückgabe:

Dokumentation der Elementfunktionen

AbstractVariableDefinition gui.datamodels.DefinitionTableDataModel.getFileAt (final int *r*)

Returns the VariableAssignment of the row r.

Parameter:

r	The row r.

Rückgabe:

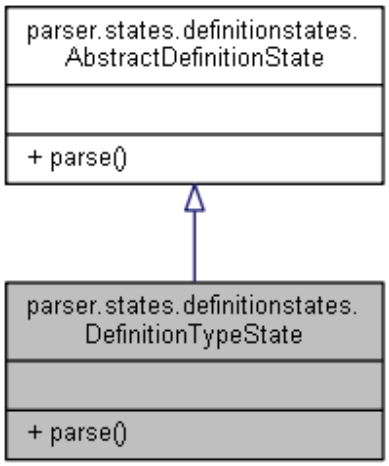
the variable assignment.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

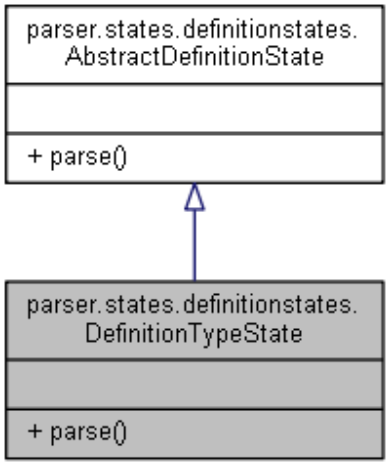
- workspace/ipfli4l13/src/gui/datamodels/DefinitionTableDataModel.java

parser.states.definitionstates.DefinitionTypeState Klassenreferenz

parser.states.definitionstates.DefinitionTypeStateKlassendiagramm für parser.states.definitionstates.DefinitionTypeState:



Zusammengehörigkeiten von parser.states.definitionstates.DefinitionTypeState:



Öffentliche Methoden

- void parse (final TempVariableDefinition tVD, final Buffer< AbstractSymbol > buffer, final DefinitionFileParser dFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.definitionstates.DefinitionTypeState.parse (final TempVariableDefinition *tVD*, final Buffer< AbstractSymbol > *buffer*, final DefinitionFileParser *dFP*) throws InterruptedException, ParserException

Generates Anonymous VariableType if a RegEx has been delivered. Otherwise, it gets the VariableTypes List and searches for it.

Parameter:

tVD	TempVariableDefinition
dFP	DefinitionFileParser
buffer	Buffer<Symbol>

Rückgabe:

Ausnahmebehandlung:

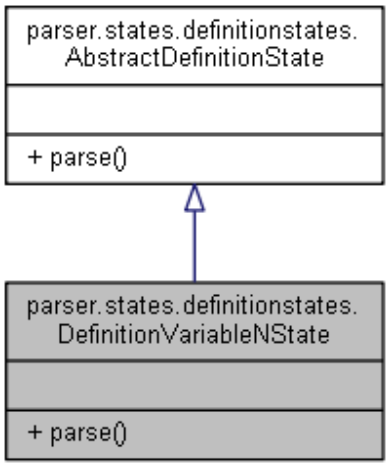
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

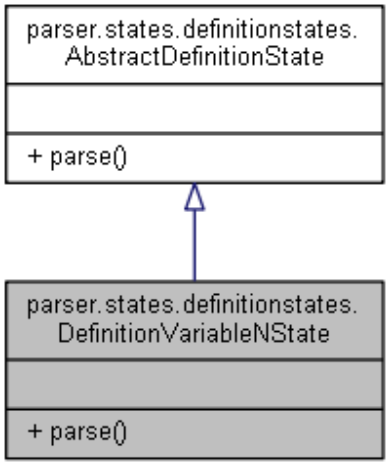
- workspace/ipfli4l13/src/parser/states/definitionstates/DefinitionTypeState.java

parser.states.definitionstates.DefinitionVariableNState Klassenreferenz

parser.states.definitionstates.DefinitionVariableNStateKlassendiagramm für parser.states.definitionstates.DefinitionVariableNState:



Zusammengehörigkeiten von parser.states.definitionstates.DefinitionVariableNState:



Öffentliche Methoden

- void parse (final TempVariableDefinition tVD, final Buffer< AbstractSymbol > buffer, final DefinitionFileParser dFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.definitionstates.DefinitionVariableNState.parse (final TempVariableDefinition *tVD*, final Buffer< AbstractSymbol > *buffer*, final DefinitionFileParser *dFP*) throws InterruptedException, ParserException

Parameter:

tVD	TempVariableDefinition
dFP	DefinitionFileParser
buffer	Buffer<Symbol>

Rückgabe:

Ausnahmebehandlung:

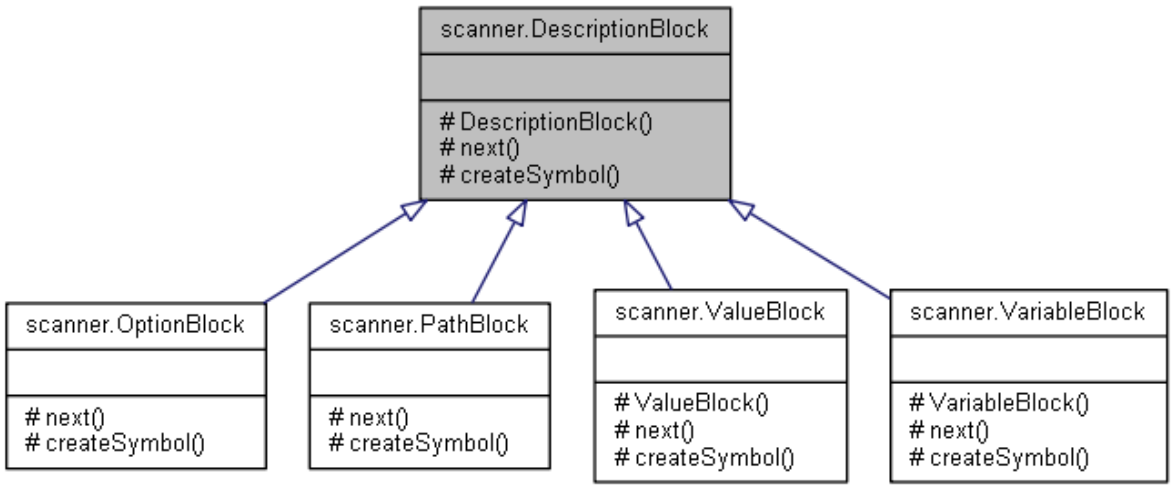
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

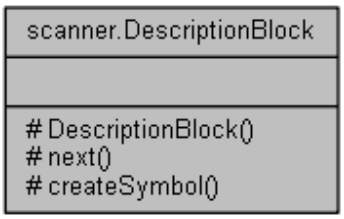
- workspace/ipfli4l13/src/parser/states/definitionstates/DefinitionVariableNState.java

scanner.DescriptionBlock Klassenreferenz

scanner.DescriptionBlockKlassendiagramm für scanner.DescriptionBlock:



Zusammengehörigkeiten von scanner.DescriptionBlock:



Geschützte Methoden

- DescriptionBlock ()
- abstract DescriptionBlock next ()
- abstract AbstractSymbol createSymbol (final String collectedContent, final Position position)

Ausführliche Beschreibung

This abstract class represents the block of the PackageDescriptionFile (The blocks are variables, values, paths and options).

Autor:

Hendrik Seffer

Beschreibung der Konstruktoren und Destruktoren

scanner.DescriptionBlock.DescriptionBlock ()`[protected]`

constructor.

Dokumentation der Elementfunktionen

abstract AbstractSymbol scanner.DescriptionBlock.createSymbol (final String *collectedContent*, final Position *position*)`[protected]`, `[pure virtual]`

Create the new Symbol.

Parameter:

collectedContent	collected Content which is used for the Symbol.
position	the position of the collected content.

Rückgabe:

the new Symbol

Implementiert in scanner.ValueBlock, scanner.VariableBlock, scanner.OptionBlock und scanner.PathBlock.

abstract DescriptionBlock scanner.DescriptionBlock.next ()`[protected]`, `[pure virtual]`

set the next State.

Rückgabe:

the new State

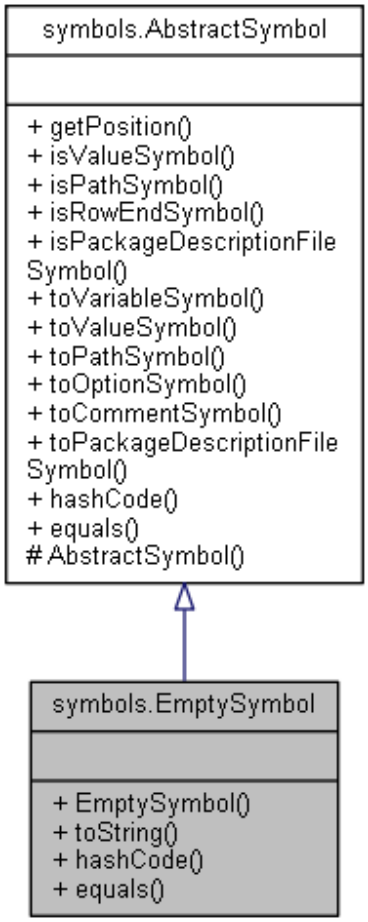
Implementiert in scanner.ValueBlock, scanner.VariableBlock, scanner.OptionBlock und scanner.PathBlock.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

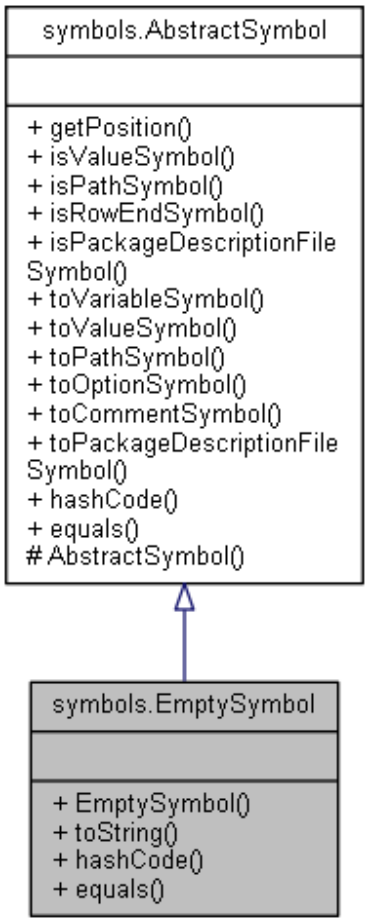
- workspace/ipfli4l13/src/scanner/DescriptionBlock.java

symbols.EmptySymbol Klassenreferenz

symbols.EmptySymbolKlassendiagramm für symbols.EmptySymbol:



Zusammengehörigkeiten von symbols.EmptySymbol:



Öffentliche Methoden

- EmptySymbol (final Position position)
- String **toString** ()
- int **hashCode** ()
- boolean **equals** (final Object obj)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the empty symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.EmptySymbol.EmptySymbol (final Position *position*)

This constructor instantiate a empty symbol.

Parameter:

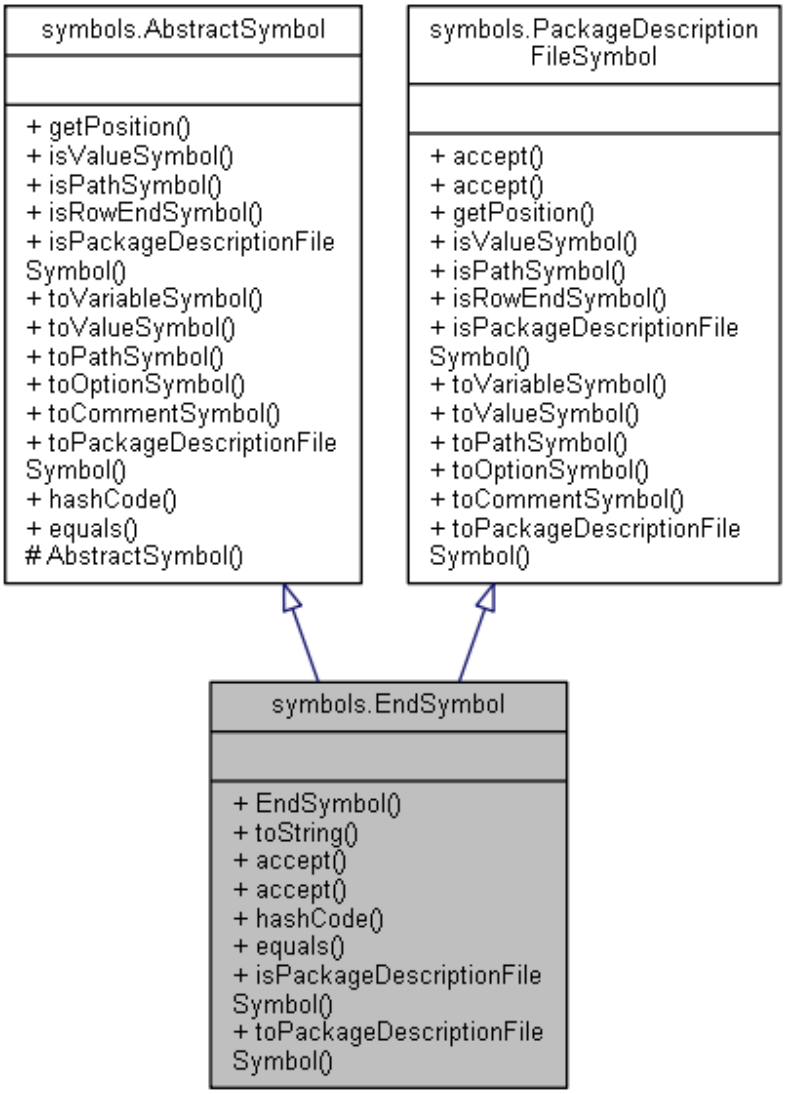
position	is the position of the data in the stream.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

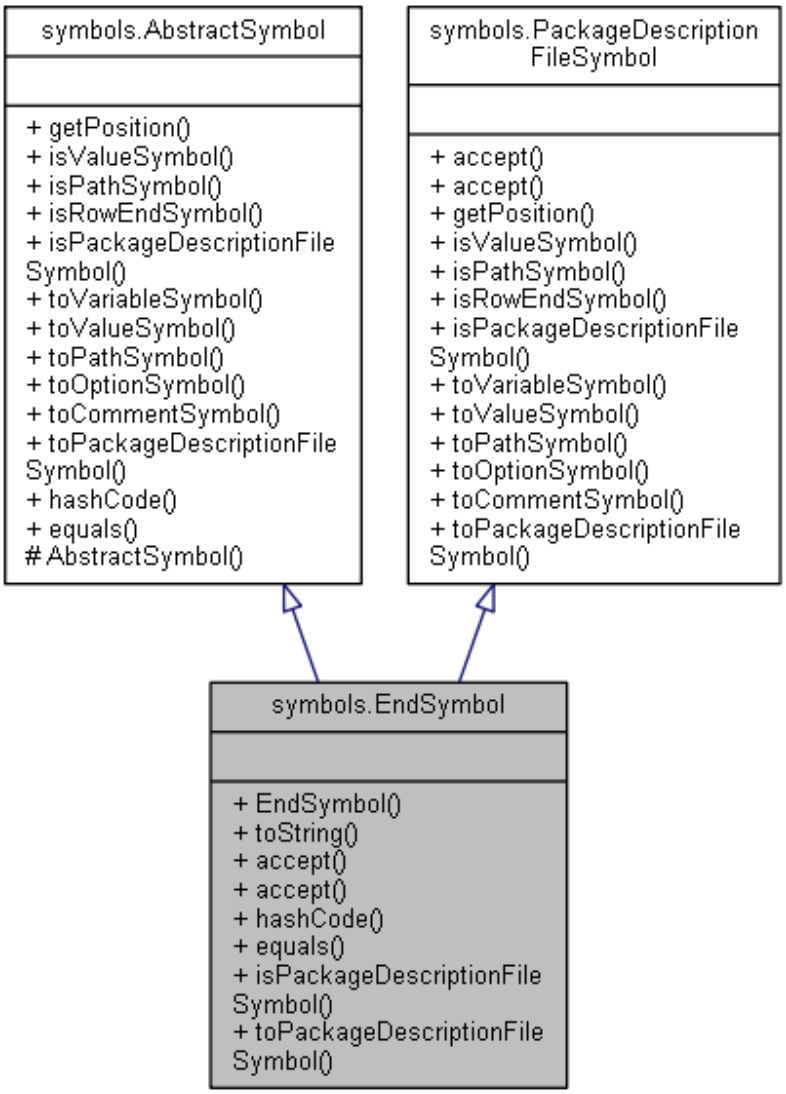
- workspace/ipfli4l13/src/symbols/EmptySymbol.java

symbols.EndSymbol Klassenreferenz

symbols.EndSymbolKlassendiagramm für symbols.EndSymbol:



Zusammengehörigkeiten von symbols.EndSymbol:



Öffentliche Methoden

- EndSymbol (final Position position)
- String **toString** ()
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitor visitor, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitorWithTempRule visitor, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- int **hashCode** ()
- boolean **equals** (final Object obj)
- Boolean isPackageDescriptionFileSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the end symbol. This is the end of the stream.

Beschreibung der Konstruktoren und Destruktoren

symbols.EndSymbol.EndSymbol (final Position *position*)

This is the constructor.

Parameter:

position	is the position of the data in the stream.

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.EndSymbol.accept (final PackageDescriptionSymbolVisitor *visitor*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

AbstractPackageParserState symbols.EndSymbol.accept (final PackageDescriptionSymbolVisitorWithTempRule *visitor*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

tempPackageRule	is a temporary package rule.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

Boolean symbols.EndSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

PackageDescriptionFileSymbol symbols.EndSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:

a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.

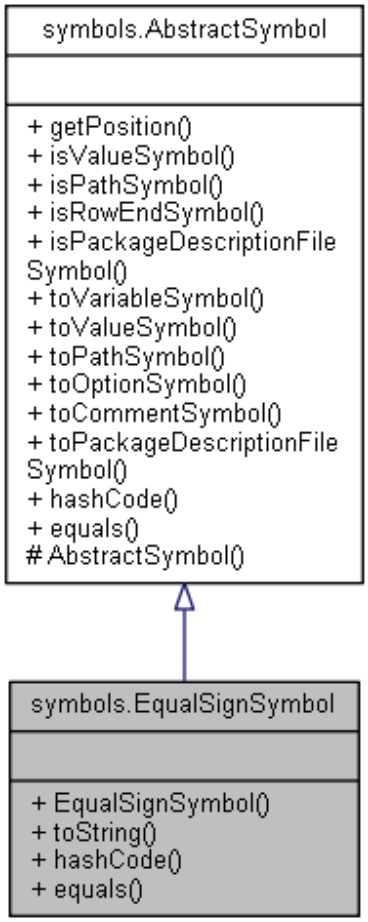
Implementiert symbols.PackageDescriptionFileSymbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

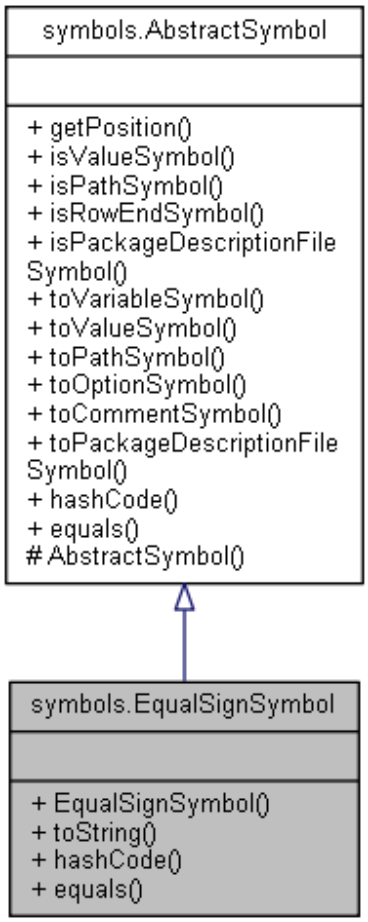
- workspace/ipfli4l13/src/symbols/EndSymbol.java

symbols.EqualSignSymbol Klassenreferenz

symbols.EqualSignSymbolKlassendiagramm für symbols.EqualSignSymbol:



Zusammengehörigkeiten von symbols.EqualSignSymbol:



Öffentliche Methoden

- EqualSignSymbol (final Position position)
- String **toString** ()
- int **hashCode** ()
- boolean **equals** (final Object obj)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the equalsign.

Beschreibung der Konstruktoren und Destruktoren

symbols.EqualSignSymbol.EqualSignSymbol (final Position *position*)

This is the constructor of the equalsign symbol.

Parameter:

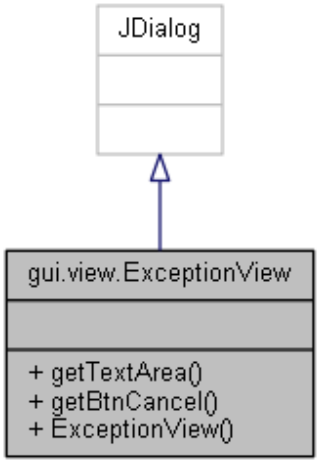
position	is the position of the data in the stream.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

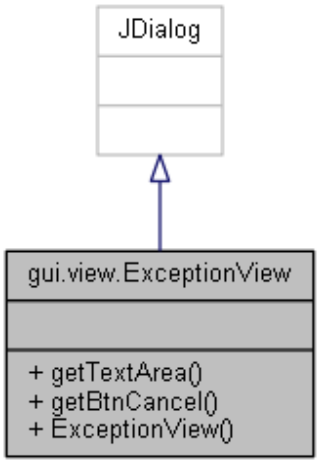
- workspace/ipfli4l13/src/symbols/EqualSignSymbol.java

gui.view.ExceptionView Klassenreferenz

gui.view.ExceptionViewKlassendiagramm für gui.view.ExceptionView:



Zusammengehörigkeiten von gui.view.ExceptionView:



Öffentliche Methoden

- final JTextArea getTextArea ()
- final JButton getBtnCancel ()
- ExceptionView (final Exception exception)

Ausführliche Beschreibung

The ExceptionView.

Autor:

Lukas

Beschreibung der Konstruktoren und Destruktoren

gui.view.ExceptionView.ExceptionView (final Exception *exception*)

Launch the application. Create the dialog.

Parameter:

exception	The exception to show.

Dokumentation der Elementfunktionen

final JButton gui.view.ExceptionView.getBtnCancel ()

Rückgabe:

the btnCancel

final JTextArea gui.view.ExceptionView.getTextArea ()

Rückgabe:

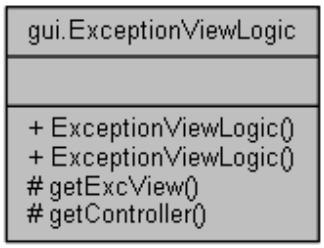
the textArea

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/view/ExceptionView.java

gui.ExceptionViewLogic Klassenreferenz

gui.ExceptionViewLogicZusammengehörigkeiten von gui.ExceptionViewLogic:



Öffentliche Methoden

- ExceptionViewLogic (final MainController controller, final Exception e)
- ExceptionViewLogic (final Exception e)

Geschützte Methoden

- ExceptionView getExcView ()
- MainController getController ()

Ausführliche Beschreibung

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.ExceptionViewLogic.ExceptionViewLogic (final MainController *controller*, final Exception *e*)

constructor. Creates the view.

Parameter:

controller	the controller to set.
e	exception to handle.

gui.ExceptionViewLogic.ExceptionViewLogic (final Exception *e*)

Constructor, creates the exception view. use this if the controller is not necessary.

Parameter:

e	The exception.

Dokumentation der Elementfunktionen

MainController gui.ExceptionViewLogic.getController ()[protected]

Rückgabe:
the controller

ExceptionView gui.ExceptionViewLogic.getExcView ()[protected]

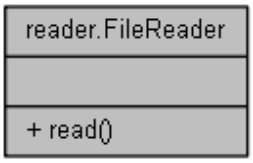
Rückgabe:
the excView

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/ExceptionViewLogic.java

reader.FileReader Klassenreferenz

reader.FileReaderZusammengehörigkeiten von reader.FileReader:



Öffentliche, statische Methoden

- static String read (final String pathToRead) throws ReaderException

---

Ausführliche Beschreibung

Autor:

Phil

Dokumentation der Elementfunktionen

static String reader.FileReader.read (final String *pathToRead*) throws ReaderException `[static]`

Reads the file in the given part and returns a string representation.

Parameter:

pathToRead	The path to read.

Ausnahmebehandlung:

ReaderException	If the Reader causes an exception.

Rückgabe:

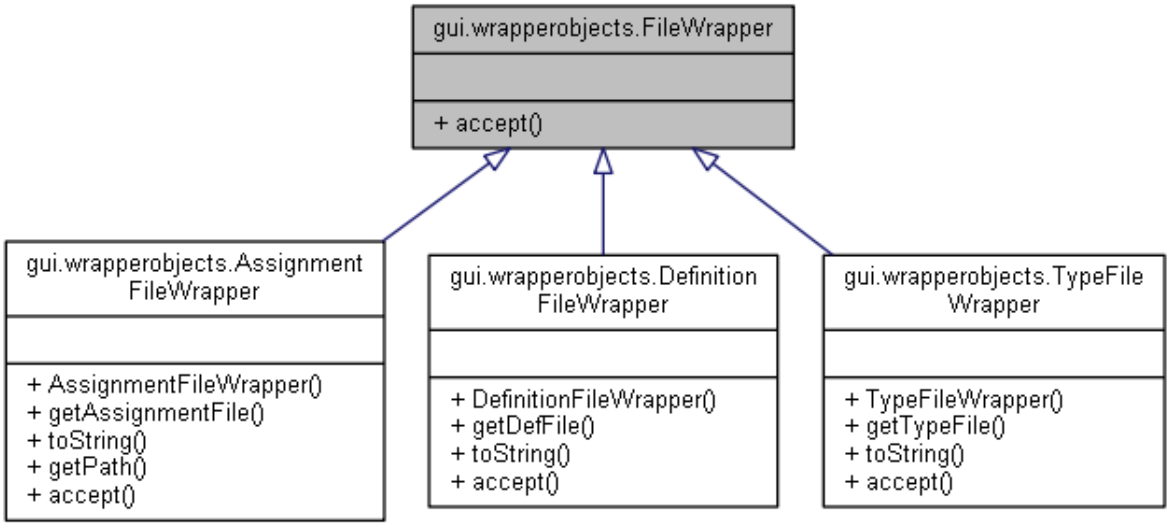
Returns a string representation of the file.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

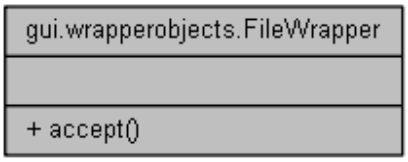
- workspace/ipfli4l13/src/reader/FileReader.java

gui.wrapperobjects.FileWrapper Schnittstellenreferenz

gui.wrapperobjects.FileWrapperKlassendiagramm für gui.wrapperobjects.FileWrapper:



Zusammengehörigkeiten von gui.wrapperobjects.FileWrapper:



Öffentliche Methoden

- void accept (final FileWrapperVisitor vis)

Ausführliche Beschreibung

interface for filewrapper.

Autor:

Phil

Dokumentation der Elementfunktionen

void gui.wrapperobjects.FileWrapper.accept (final FileWrapperVisitor *vis*)

accept-method for visitor pattern.

Parameter:

vis	the visitor.

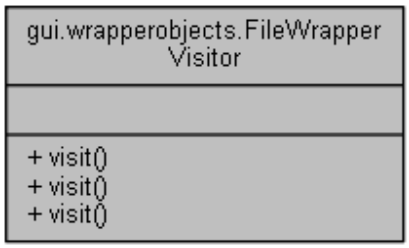
Implementiert in gui.wrapperobjects.AssignmentFileWrapper, gui.wrapperobjects.DefinitionFileWrapper und gui.wrapperobjects.TypeFileWrapper.

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/wrapperobjects/FileWrapper.java

gui.wrapperobjects.FileWrapperVisitor Schnittstellenreferenz

gui.wrapperobjects.FileWrapperVisitorZusammengehörigkeiten von gui.wrapperobjects.FileWrapperVisitor:



Öffentliche Methoden

- void visit (final AssignmentFileWrapper assFileWrap)
- void visit (final DefinitionFileWrapper defFileWrap)
- void visit (final TypeFileWrapper typeFileWrap)

Ausführliche Beschreibung

A FileWrapperVisitor.

Autor:

Phil

Dokumentation der Elementfunktionen

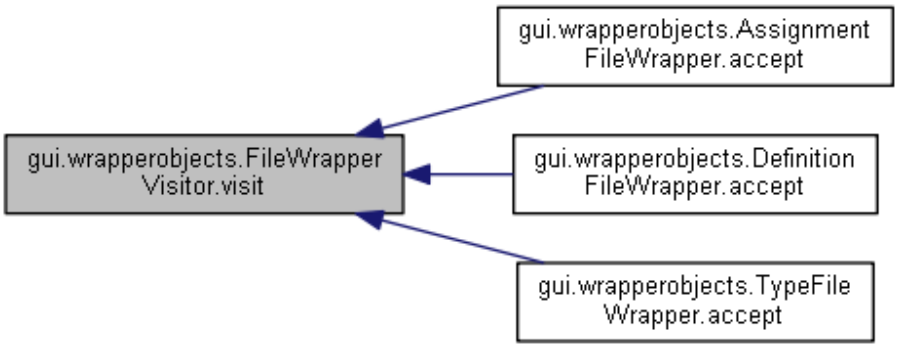
void gui.wrapperobjects.FileWrapperVisitor.visit (final AssignmentFileWrapper *assFileWrap*)

visit-method for assignmentfilewrapper.

Parameter:

assFileWrap	The file to visit.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



void gui.wrapperobjects.FileWrapperVisitor.visit (final DefinitionFileWrapper *defFileWrap*)

visit-method for definitionfilewrapper.

Parameter:

defFileWrap	The file to visit.

void gui.wrapperobjects.FileWrapperVisitor.visit (final TypeFileWrapper *typeFileWrap*)

visit-method for typefilewrapper.

Parameter:

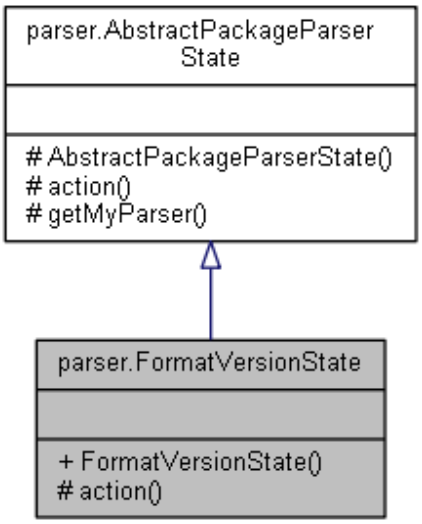
typeFileWrap	The file to visit.

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

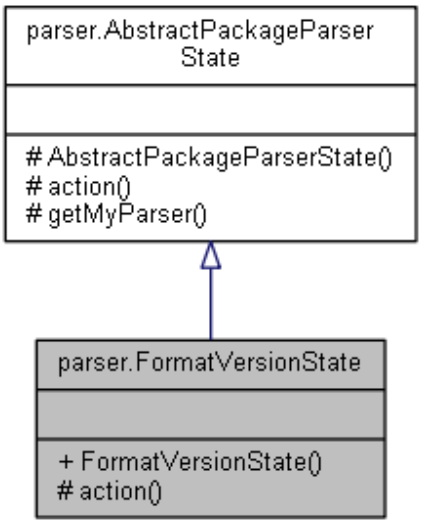
- workspace/ipfli4l13/src/gui/wrapperobjects/FileWrapperVisitor.java

parser.FormatVersionState Klassenreferenz

parser.FormatVersionStateKlassendiagramm für parser.FormatVersionState:



Zusammengehörigkeiten von parser.FormatVersionState:



Öffentliche Methoden

- FormatVersionState (final PackageDescriptionFileParser parser)

Geschützte Methoden

- AbstractPackageParserState **action** (final PackageDescriptionFileSymbol symbol, final Map< String, AbstractVariableDefinition > variableDef) throws ParserException, InterruptedException

Ausführliche Beschreibung

The class represents the FormatVersionState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.FormatVersionState.FormatVersionState (final PackageDescriptionFileParser *parser*)

constructor.

Parameter:

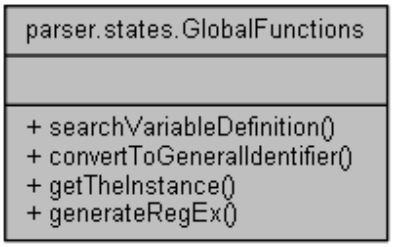
parser	for the PackageDescriptionFile.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/FormatVersionState.java

parser.states.GlobalFunctions Klassenreferenz

parser.states.GlobalFunctionsZusammengehörigkeiten von parser.states.GlobalFunctions:



Öffentliche Methoden

- AbstractVariableDefinition searchVariableDefinition (final Map< String, AbstractVariableDefinition > varDefList, final String identifier) throws ParserException
- String convertToGeneralIdentifier (final String identifier)

Öffentliche, statische Methoden

- static GlobalFunctions getTheInstance ()
- static RegEx generateRegEx (final String string)

Ausführliche Beschreibung

Autor:

Muri

Dokumentation der Elementfunktionen

String parser.states.GlobalFunctions.convertToGeneralIdentifier (final String *identifier*)

converts an array-elements identifier into its generalized name i.e.: "PF_USR_CHAIN_2_RULE_4_COMMENT" -> "PF_USR_CHAIN_%_RULE_%_COMMENT".

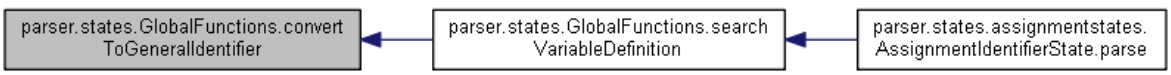
Parameter:

identifier	identifier for the element to be parsed

Rückgabe:

the generalized identifier

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



static RegEx parser.states.GlobalFunctions.generateRegEx (final String *string*)[static]

Parameter:

string	input String

Rückgabe:

ArrayList of RegExParts

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

static GlobalFunctions parser.states.GlobalFunctions.getTheInstance ()[static]

Rückgabe:

theInstance

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

AbstractVariableDefinition parser.states.GlobalFunctions.searchVariableDefinition (final Map< String, AbstractVariableDefinition > *varDefList*, final String *identifier*) throws ParserException

gets the VariableDefinition of the searched Identifier.

Parameter:

varDefList	varDefList
identifier	identifier

Rückgabe:

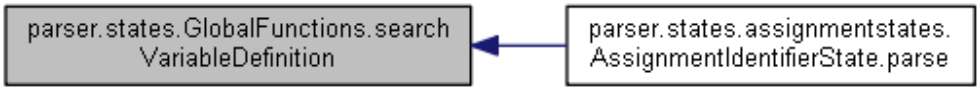
the VariableDefiniton for an Identifier

Ausnahmebehandlung:

ParserException	ParserException if VariableDefinition is not found

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/states/GlobalFunctions.java

basic.GuiConstants Klassenreferenz

basic.GuiConstantsZusammengehörigkeiten von basic.GuiConstants:

Statische öffentliche Attribute

- static final String SAVETEXT = "save"
- static final String **ROUTERCONFIG** = "Router"
- static final int **FRAMEBOUNDX** = 100
- static final int **FRAMEBOUNDY** = 100
- static final int **FRAMEBOUNDWIDTH** = 840
- static final int **FRAMEBOUNDHEIGHT** = 510
- static final int **BORDER** = 5
- static final int **NORTHPAD** = -212
- static final int **WESTPAD** = -10
- static final int **EASTPAD** = -6
- static final int **SCROLLTABNORTHPAD** = 10
- static final int **SCROLLTABEASTPAD** = -10
- static final int **SCROLLTABSOUTHPAD** = 306
- static final int **SCROLLTABWESTPAD** = 299
- static final int **SEPHOZNORTHPAD** = 45
- static final int **SEPHOZEASTPAD** = 810
- static final int **SEPHOZWESTPAD** = 10
- static final int **PADDING0** = 0
- static final int **PADDING6** = 6
- static final int **PADDING40** = 40
- static final int **PADDING61** = 61
- static final int **PADDING252** = 252
- static final int **PADDING212** = 212
- static final int **PADDINGN702** = -702
- static final int **BOUNDSX** = 100
- static final int **BOUNDSY** = 100
- static final int **BOUNDSWIDTH** = 450
- static final int **BOUNDSHEIGHT** = 300
- static final int **DIALOGBOXWIDTH** = 450
- static final int **DIALOGBOXHEIGHT** = 149
- static final int **TEXTFIELDCOLS** = 10

- static final int **TFRNNORTH** = 10
- static final int **TFRNWEST** = 92
- static final int **TFRNEAST** = 429
- static final int **BTNSPNORTH** = 36
- static final int **BTNSPWEST** = 338
- static final int **TFSPNORTH** = 37
- static final int **TFSPWEST** = 92
- static final int **TFSPEAST** = 332
- static final int **LBLSPNORTH** = 40
- static final int **LBLSPWEST** = 15
- static final int **LBLRNNORTH** = 13
- static final int **LBLRNWEST** = 15
- static final String **LOOKANDFEELEXCP** = "Unavailable Look & Feel: "
- static final String **OKAYTEXT** = "Okay"
- static final String **CANCELTEXT** = "Cancel"
- static final String **LABELTEXT** = "Label"
- static final String **ROUTERREP** = "ROUTER: "
- static final int **DEFAULTPADDING** = -10
- static final String **ADDROUTERBTNTEXT** = "Add Router"
- static final String **SAVEASSIGNBTNTEXT** = "save assignment"
- static final String **JFRAMETITLE** = "Fli4l"
- static final String **ASSIGNMENTTEXT** = "Assignment"
- static final String **ASSIGNMENTSTEXT** = "Assignments"
- static final String **TYPESTEXT** = "Types"
- static final String **DEFINITIONSTEXT** = "Definitions"
- static final String **PACKTEXT** = "Package: "
- static final String **VAROPTTEXT** = "Variable OPT"
- static final String **VARNTEXT** = "Variable N"
- static final String **DEFAULTVALUETEXT** = "Default Value"
- static final String **TYPETEXT** = "Type"
- static final String **REGEXTEXT** = "Regular Expression"
- static final String **ERRMTEXT** = "Error Message"
- static final String **SAVECONFIRMDIALOG**
- static final String **PATHNOTFOUNDEXC** = "Path not found!"
- static final String **NAMEEMPTY** = "The router name isn´t set. Pls fill the name label!"
- static final int **DEFINITIONDIVIDE** = 5
- static final int **TYPEDIVIDE** = 4
- static final int **ASSIGNDIVIDE** = 2
- static final int **DEFAULTNORTHPAD** = 50
- static final int **RIGHTPANWESTPAD** = 300
- static final int **JSEPSOUPAD** = -5
- static final String **LOOKANDFEEL** = "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel"

Ausführliche Beschreibung

UtilityClass for gui constants.

Autor:

Phil

Dokumentation der Datenelemente

final String basic.GuiConstants.SAVECONFIRMDIALOG[static]

```
Initialisierung:=
            "Would you like to save the selected assignment?"
```

final String basic.GuiConstants.SAVETEXT = "save"[static]

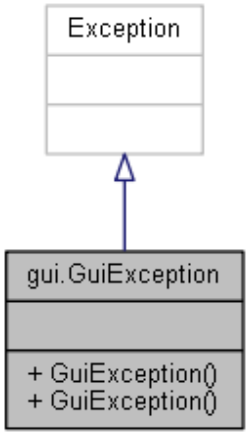
String representation for the savetext.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

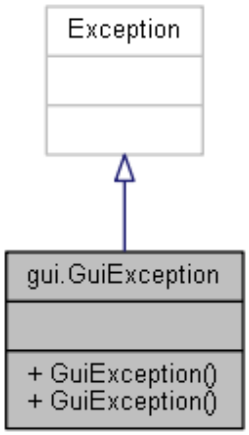
- workspace/ipfli4l13/src/basic/GuiConstants.java

gui.GuiException Klassenreferenz

gui.GuiExceptionKlassendiagramm für gui.GuiException:



Zusammengehörigkeiten von gui.GuiException:



Öffentliche Methoden

- GuiException (final String message, final Exception causeE)
- GuiException (final String message)

Ausführliche Beschreibung

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.GuiException.GuiException (final String *message*, final Exception *causeE*)

Constructor for creating an Exception for all Exceptions caused by the gui.

Parameter:

causeE	The original exception caused by java.
message	The message to set.

gui.GuiException.GuiException (final String *message*)

Constructor for creating an Exception for all Exceptions caused by the gui.

Parameter:

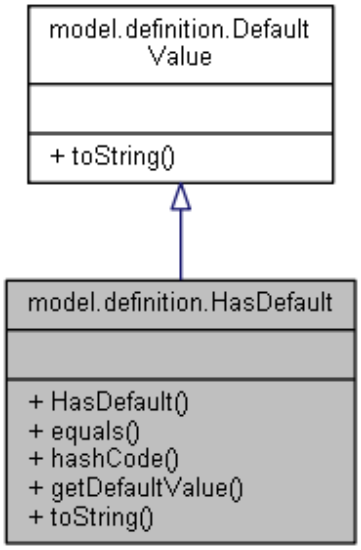
message	The message to set.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

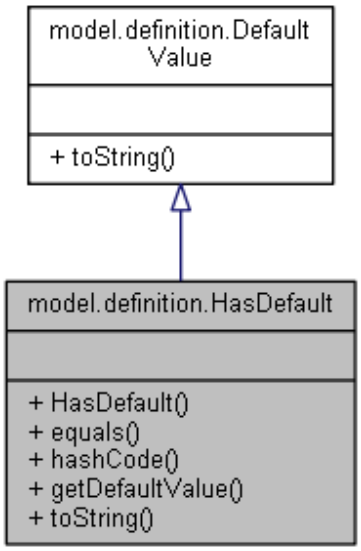
- workspace/ipfli4l13/src/gui/GuiException.java

model.definition.HasDefault Klassenreferenz

model.definition.HasDefaultKlassendiagramm für model.definition.HasDefault:



Zusammengehörigkeiten von model.definition.HasDefault:



Öffentliche Methoden

- HasDefault (final String defaultValue)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- String getDefaultValue ()
- String **toString** ()

Ausführliche Beschreibung

Representation for definitions with a default value.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.definition.HasDefault.HasDefault (final String *defaultValue*)

Constructor, just sets fields. No side effects.

Parameter:

defaultValue	The default value to set.

Dokumentation der Elementfunktionen

String model.definition.HasDefault.getDefaultValue ()

Returns the field defaultValue. No side effects.

Rückgabe:

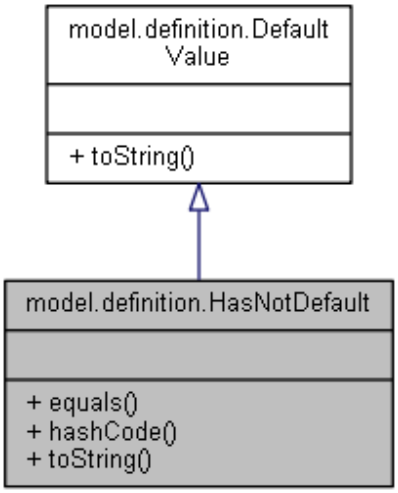
this.defaultValue;

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

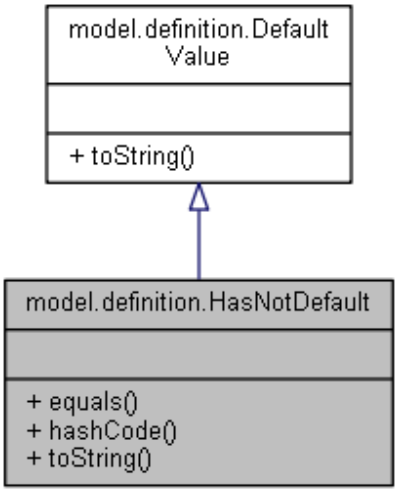
- workspace/ipfli4l13/src/model/definition/HasDefault.java

model.definition.HasNotDefault Klassenreferenz

model.definition.HasNotDefaultKlassendiagramm für model.definition.HasNotDefault:



Zusammengehörigkeiten von model.definition.HasNotDefault:



Öffentliche Methoden

- boolean **equals** (final Object obj)
- int **hashCode** ()
- String **toString** ()

Ausführliche Beschreibung

Representation for definitions without a default value.

Autor:

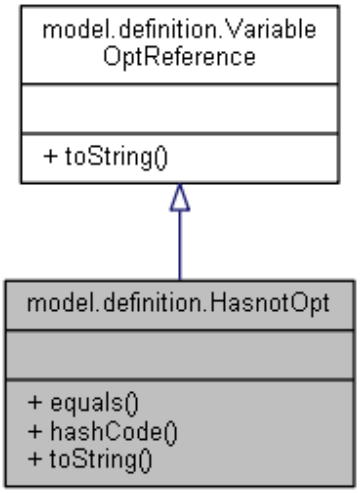
Represents value of the variable.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

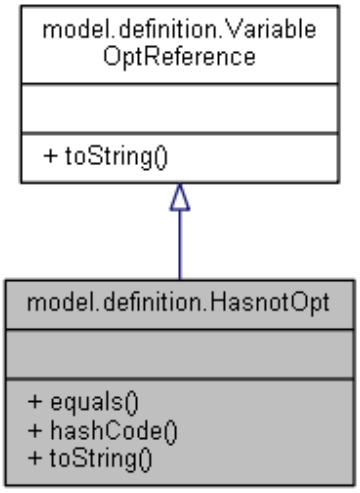
- workspace/ipfli4l13/src/model/definition/HasNotDefault.java

model.definition.HasnotOpt Klassenreferenz

model.definition.HasnotOptKlassendiagramm für model.definition.HasnotOpt:



Zusammengehörigkeiten von model.definition.HasnotOpt:



Öffentliche Methoden

- boolean **equals** (final Object obj)
- int **hashCode** ()
- String **toString** ()

Ausführliche Beschreibung

Representation for definitions without an Opt.

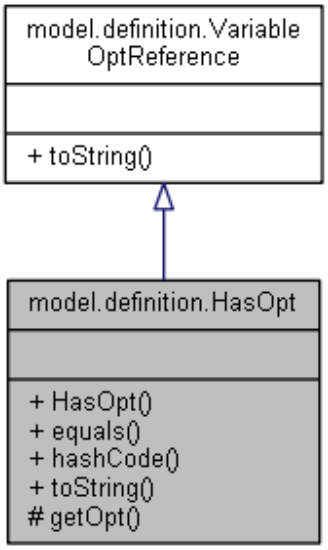
Autor:
HFW410RA - Philipp Rammos

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

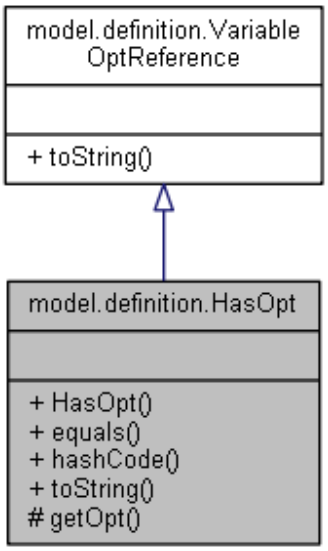
- workspace/ipfli4l13/src/model/definition/HasnotOpt.java

model.definition.HasOpt Klassenreferenz

model.definition.HasOptKlassendiagramm für model.definition.HasOpt:



Zusammengehörigkeiten von model.definition.HasOpt:



Öffentliche Methoden

- HasOpt (final Opt opt)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- String **toString** ()

Geschützte Methoden

- Opt getOpt ()

Ausführliche Beschreibung

Representation for definitions with an Opt.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.definition.HasOpt.HasOpt (final Opt *opt*)

Constructor, just sets the field. No side effects.

Parameter:

opt	The Opt to set.

Dokumentation der Elementfunktionen

Opt model.definition.HasOpt.getOpt ()[protected]

Returns the field opt. No side effects.

Rückgabe:

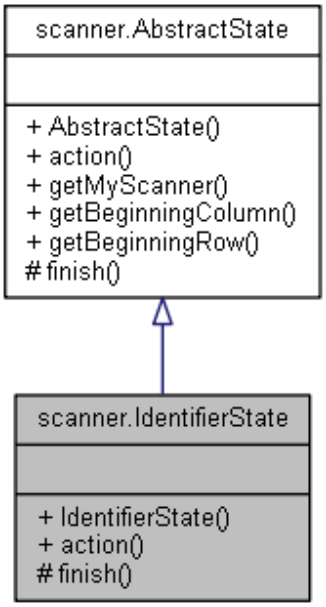
this.opt

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

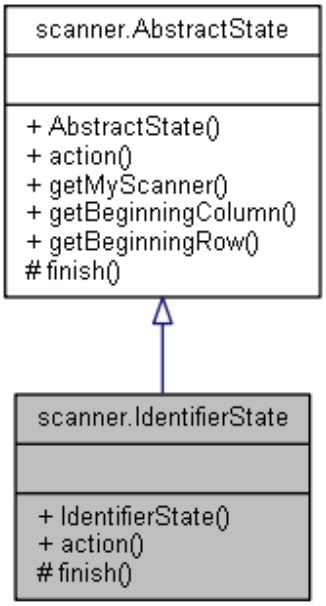
- workspace/ipfli4l13/src/model/definition/HasOpt.java

scanner.IdentifierState Klassenreferenz

scanner.IdentifierStateKlassendiagramm für scanner.IdentifierState:



Zusammengehörigkeiten von scanner.IdentifierState:



Öffentliche Methoden

- IdentifierState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow, final String collectedIdentifier)
- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

This class represents the IdentifierState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.IdentifierState.IdentifierState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*, final String *collectedIdentifier*)

This constructor is used to initialize the IdentifierState.

Parameter:

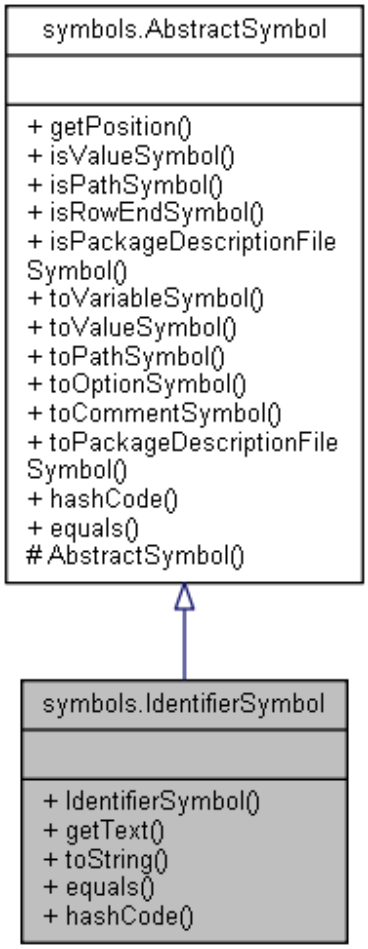
myScanner	is the used scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the beginning Row of the Symbol
collectedIdentifier	the collected Identifier Character as String.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

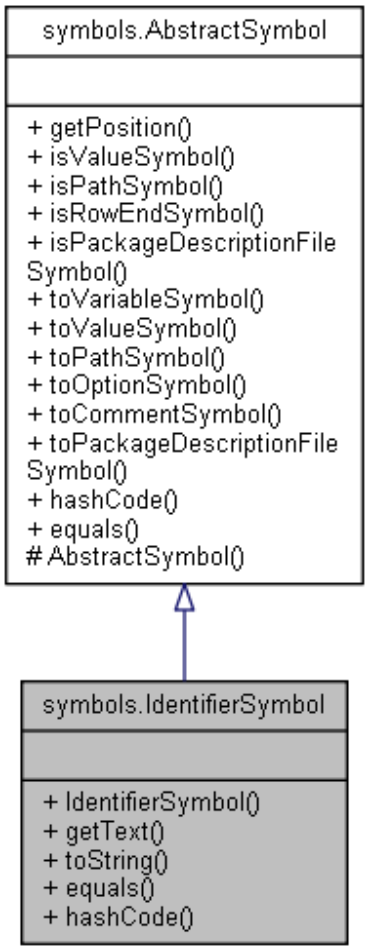
- workspace/ipfli4l13/src/scanner/IdentifierState.java

symbols.IdentifierSymbol Klassenreferenz

symbols.IdentifierSymbolKlassendiagramm für symbols.IdentifierSymbol:



Zusammengehörigkeiten von symbols.IdentifierSymbol:



Öffentliche Methoden

- IdentifierSymbol (final String text, final Position position)
- String getText ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the Identifier Symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.IdentifierSymbol.IdentifierSymbol (final String *text*, final Position *position*)

The constructor instantiate the identifier with a

Parameter:

text.	
text	is the text of the identifier.
position	is the position of the data in the stream.

Dokumentation der Elementfunktionen

String symbols.IdentifierSymbol.getText ()

This method

Rückgabe:

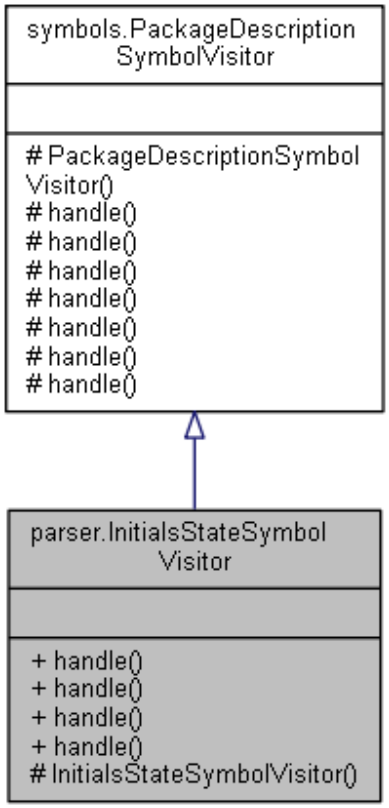
the text of the identifier.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

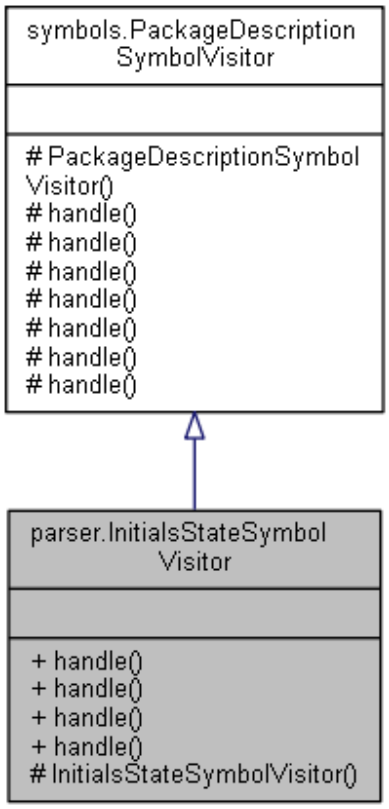
- workspace/ipfli4l13/src/symbols/IdentifierSymbol.java

parser.InitialsStateSymbolVisitor Klassenreferenz

parser.InitialsStateSymbolVisitorKlassendiagramm für parser.InitialsStateSymbolVisitor:



Zusammengehörigkeiten von parser.InitialsStateSymbolVisitor:



Öffentliche Methoden

- AbstractPackageParserState **handle** (final CommentSymbol commentSymbol, final PackageDescriptionFileParser parser)
- AbstractPackageParserState **handle** (final EndSymbol endSymbol, final PackageDescriptionFileParser parser) throws InterruptedException
- AbstractPackageParserState **handle** (final RowEndSymbol rowEndSymbol, final PackageDescriptionFileParser parser) throws InterruptedException
- AbstractPackageParserState **handle** (final VariableSymbol variableSymbol, final PackageDescriptionFileParser parser)

Geschützte Methoden

- InitialsStateSymbolVisitor ()

Ausführliche Beschreibung

This class represents the Visitor for the InitialStateSymbol.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.InitialsStateSymbolVisitor.InitialsStateSymbolVisitor ()[protected]

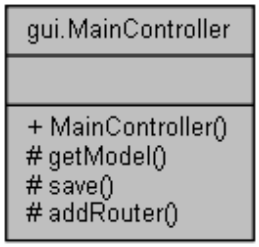
creates a InitialStateSymbolVisitor.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/InitialsStateSymbolVisitor.java

gui.MainController Klassenreferenz

gui.MainControllerZusammengehörigkeiten von gui.MainController:



Öffentliche Methoden

- MainController (final ModelFacade modelFacade)

Geschützte Methoden

- ModelFacade getModel ()
- void save (final AssignmentFile assignFile, final String path) throws PrinterException
- List< model.Package > addRouter (final String name, final String pathToFli4l) throws ScannerException, InterruptedException, IOException, ReaderException, ParserException

---

Ausführliche Beschreibung

The controller (MVC-Pattern). Controls between ViewLogic and (Business/Core)Model.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.MainController.MainController (final ModelFacade *modelFacade*)

Constructor, sets fields.

Parameter:

modelFacade	the model to set.

Dokumentation der Elementfunktionen

List<model.Package> gui.MainController.addRouter (final String *name*, final String *pathToFli4l*) throws ScannerException, InterruptedException, IOException, ReaderException, ParserException `[protected]`

Parameter:

pathToFli4l	The path to fli4l.
name	The name of the router to set.

Rückgabe:

Returns a list of completely scanned and parsed packages.

Ausnahmebehandlung:

ParserException	If the parser causes an exception.
ReaderException	If the reader causes an exception.
IOException	If the scanner causes an exception.
InterruptedException	If the scanner causes an exception.
ScannerException	If the scanner causes an exception.

ModelFacade gui.MainController.getModel ()[protected]

Rückgabe:

the coreModel

void gui.MainController.save (final AssignmentFile *assignFile*, final String *path*) throws PrinterException [protected]

Parameter:

path	The path for printing.
assignFile	The assignmentfile to print/save.

Ausnahmebehandlung:

PrinterException	If the printer causes an exception.

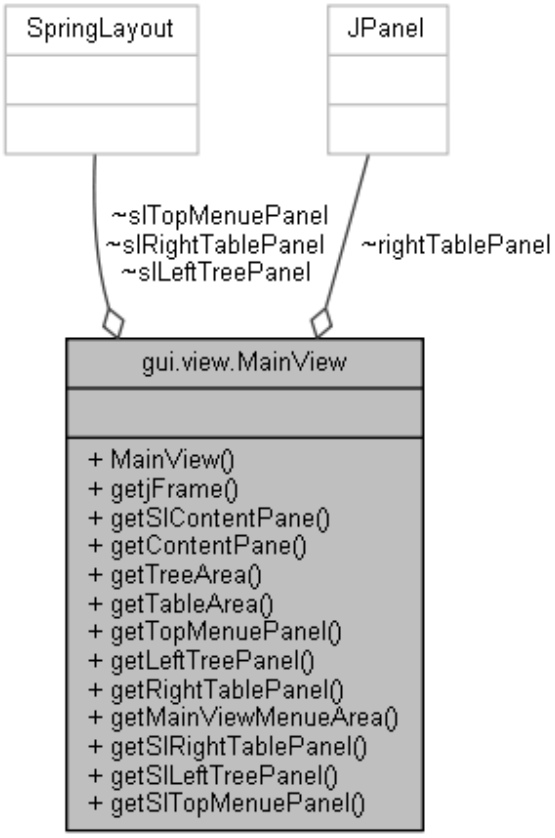
Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/MainController.java

gui.view.MainView Klassenreferenz

gui.view.MainViewZusammengehörigkeiten von gui.view.MainView:



Öffentliche Methoden

- MainView ()
- JFrame getjFrame ()
- SpringLayout getSlContentPane ()
- JPanel getContentPane ()
- MainViewTreeArea getTreeArea ()
- MainViewTableArea getTableArea ()
- JPanel getTopMenuePanel ()
- JPanel getLeftTreePanel ()
- JPanel getRightTablePanel ()
- MainViewMenueArea getMainViewMenueArea ()
- SpringLayout getSlRightTablePanel ()
- SpringLayout getSlLeftTreePanel ()
- SpringLayout getSlTopMenuePanel ()

Ausführliche Beschreibung

The Main-View.

Autor:
Lukas/Mark

Beschreibung der Konstruktoren und Destruktoren

gui.view.MainView.MainView ()

Create the frame.

Dokumentation der Elementfunktionen

JPanel gui.view.MainView.getContentPane ()

Rückgabe:
the contentPane

JFrame gui.view.MainView.getjFrame ()

Rückgabe:
the jFrame

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

JPanel gui.view.MainView.getLeftTreePanel ()

Rückgabe:
the leftTreePanel

MainViewMenueArea gui.view.MainView.getMainViewMenueArea ()

Rückgabe:
the mainViewMenueArea

JPanel gui.view.MainView.getRightTablePanel ()

Rückgabe:
the rightTablePanel

SpringLayout gui.view.MainView.getSlContentPane ()

Rückgabe:
the slContentPane

SpringLayout gui.view.MainView.getSlLeftTreePanel ()

Rückgabe:
the slLeftTreePanel

SpringLayout gui.view.MainView.getSlRightTablePanel ()

Rückgabe:
the slRightTablePanel

SpringLayout gui.view.MainView.getSlTopMenuePanel ()

Rückgabe:
the slTopMenuePanel

MainViewTableArea gui.view.MainView.getTableArea ()

Rückgabe:
the tableArea

JPanel gui.view.MainView.getTopMenuePanel ()

Rückgabe:
the topMenuePanel

MainViewTreeArea gui.view.MainView.getTreeArea ()

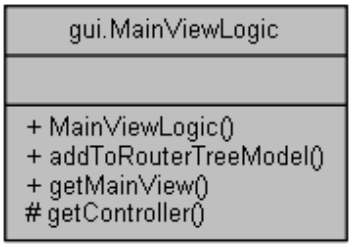
Rückgabe:
the treeArea

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/view/MainView.java

gui.MainViewLogic Klassenreferenz

gui.MainViewLogicZusammengehörigkeiten von gui.MainViewLogic:



Öffentliche Methoden

- MainViewLogic (final MainController controller)
- void addToRouterTreeModel (final List< Package > router, final String routerName)
- MainView getMainView ()

Geschützte Methoden

- MainController getController ()

Ausführliche Beschreibung

Contains model and logic for GUI-elements.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.MainViewLogic.MainViewLogic (final MainController *controller*)

constructor. Creates the view.

Parameter:

controller	the controller to set.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Dokumentation der Elementfunktionen

void gui.MainViewLogic.addToRouterTreeModel (final List< Package > *router*, final String *routerName*)

Parameter:

router	A list of the packages.
routerName	The name of the router.

MainController gui.MainViewLogic.getController ()[protected]

Rückgabe:
the controller

MainView gui.MainViewLogic.getMainView ()

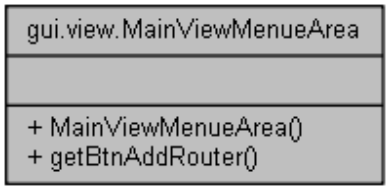
Rückgabe:
the view

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/MainViewLogic.java

gui.view.MainViewMenueArea Klassenreferenz

gui.view.MainViewMenueAreaZusammengehörigkeiten von gui.view.MainViewMenueArea:



Öffentliche Methoden

- MainViewMenueArea (final MainView mainView)
- JButton getBtnAddRouter ()

Ausführliche Beschreibung

The top menue area of the main view.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.view.MainViewMenueArea.MainViewMenueArea (final MainView *mainView*)

Parameter:

mainView	The mainview to set.

Dokumentation der Elementfunktionen

JButton gui.view.MainViewMenueArea.getBtnAddRouter ()

Rückgabe:

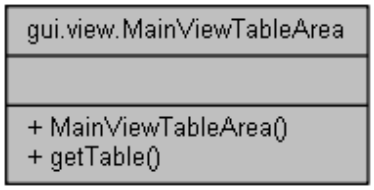
the btnAddRouter

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/view/MainViewMenueArea.java

gui.view.MainViewTableArea Klassenreferenz

gui.view.MainViewTableAreaZusammengehörigkeiten von gui.view.MainViewTableArea:



Öffentliche Methoden

- MainViewTableArea (final MainView mainView)
- JTable getTable ()

Ausführliche Beschreibung

The table area of the mainview.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.view.MainViewTableArea.MainViewTableArea (final MainView *mainView*)

Parameter:

mainView	The mainview to set.

Dokumentation der Elementfunktionen

JTable gui.view.MainViewTableArea.getTable ()

Rückgabe:

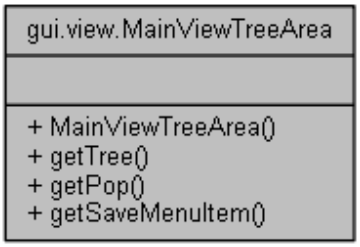
the table

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/view/MainViewTableArea.java

gui.view.MainViewTreeArea Klassenreferenz

gui.view.MainViewTreeAreaZusammengehörigkeiten von gui.view.MainViewTreeArea:



Öffentliche Methoden

- MainViewTreeArea (final MainView mainView)
- JTree getTree ()
- JPopupMenu getPop ()
- JMenuItem getSaveMenuItem ()

Ausführliche Beschreibung

The tree area of the mainview.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.view.MainViewTreeArea.MainViewTreeArea (final MainView *mainView*)

Parameter:

mainView	The mainview to set.

Dokumentation der Elementfunktionen

JPopupMenu gui.view.MainViewTreeArea.getPop ()

Rückgabe:

the pop

JMenuItem gui.view.MainViewTreeArea.getSaveMenuItem ()

Rückgabe:

the saveMenuItem

JTree gui.view.MainViewTreeArea.getTree ()

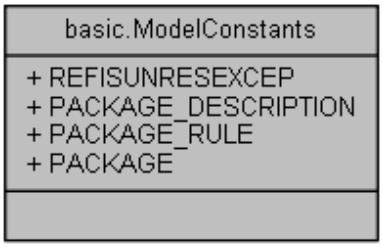
Rückgabe:
the tree

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/view/MainViewTreeArea.java

basic.ModelConstants Klassenreferenz

basic.ModelConstantsZusammengehörigkeiten von basic.ModelConstants:



Statische öffentliche Attribute

- static final String REFISUNRESEXCEP
- static final String PACKAGE_DESCRIPTION = "Package description: "
- static final String PACKAGE_RULE = "Package rule: "
- static final String PACKAGE = "package: "

Ausführliche Beschreibung

UtilityClass for model constants.

Autor:

Phil

Dokumentation der Datenelemente

final String basic.ModelConstants.PACKAGE = "package: "[static]

String representation for package rule.

final String basic.ModelConstants.PACKAGE_DESCRIPTION = "Package description: "[static]

String representation for package description.

final String basic.ModelConstants.PACKAGE_RULE = "Package rule: "[static]

String representation for package rule.

final String basic.ModelConstants.REFISUNRESEXCEP[static]

```
Initialisierung:=
            "A part of the regular expression is unresolved: "
```

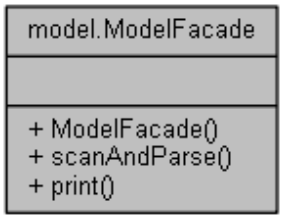
String representation for ReferenceIsUnresolvedException.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/basic/ModelConstants.java

model.ModelFacade Klassenreferenz

model.ModelFacadeZusammengehörigkeiten von model.ModelFacade:



Öffentliche Methoden

- ModelFacade ()
- model.Package scanAndParse (final File assignFile, final File defFile, final File typeFile, final File packDescFile, final String name) throws ScannerException, InterruptedException, IOException, ReaderException, ParserException
- void print (final AssignmentFile assignFile, final String path) throws PrinterException

---

Ausführliche Beschreibung

A Facade to the model.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

model.ModelFacade.ModelFacade ()

Constructor for ModelFacade, just sets the fields.

Dokumentation der Elementfunktionen

void model.ModelFacade.print (final AssignmentFile *assignFile*, final String *path*) throws PrinterException

Parameter:

path	The path for printing.
assignFile	The assignmentfile to print/save.

Ausnahmebehandlung:

PrinterException	If the printer causes an exception.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

model.Package model.ModelFacade.scanAndParse (final File *assignFile*, final File *defFile*, final File *typeFile*, final File *packDescFile*, final String *name*) throws ScannerException, InterruptedException, IOException, ReaderException, ParserException

Parameter:

assignFile	The assignmentfile to scan and parse.
defFile	The definitionfile to scan and parse.
typeFile	The typeFile to scan and parse.
name	The name of the package.
packDescFile	The descriptionfile to set.

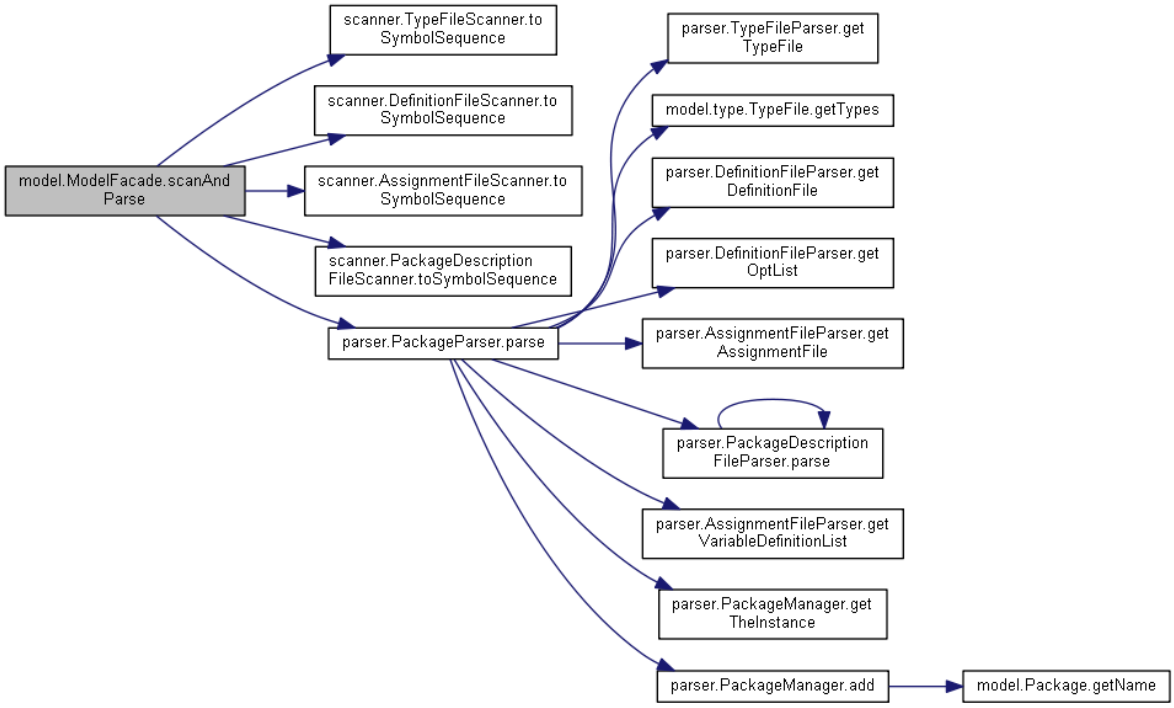
Ausnahmebehandlung:

ReaderException	If the reader throws an exception.
IOException	If the scanner throws an exception.
InterruptedException	If the scanner throws an exception.
ScannerException	If the scanner throws an exception.
ParserException	If the parser throws an exception.

Rückgabe:

Returns the complete package.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

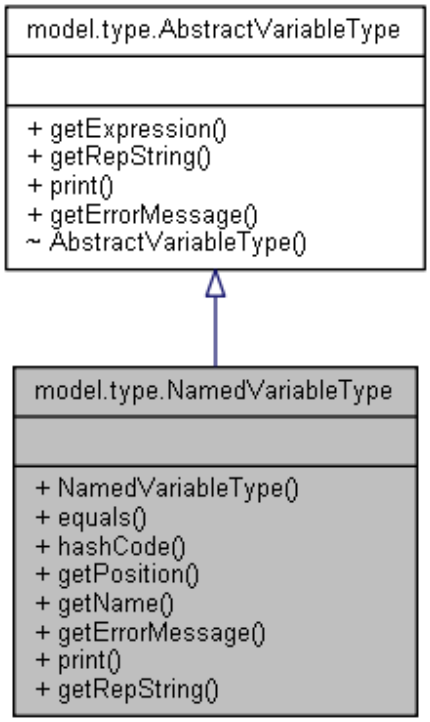


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

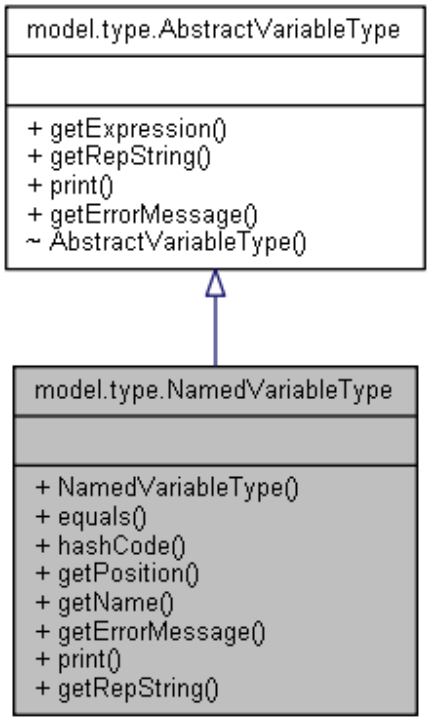
- workspace/ipfli4l13/src/model/ModelFacade.java

model.type.NamedVariableType Klassenreferenz

model.type.NamedVariableTypeKlassendiagramm für model.type.NamedVariableType:



Zusammengehörigkeiten von model.type.NamedVariableType:



Öffentliche Methoden

- NamedVariableType (final String name, final RegEx expression, final String errorMessage, final Position position)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- Position getPosition ()
- String getName ()
- String getErrorMessage ()
- String print ()
- String getRepString ()

Ausführliche Beschreibung

Represents a type of an definition. An entry from the TypeFile.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.NamedVariableType.NamedVariableType (final String *name*, final RegEx *expression*, final String *errorMessage*, final Position *position*)

Constructor, just sets the fields. No side effects.

Parameter:

name	The name to set.
expression	the expression to set.
errorMessage	the error message to set
position	The psition to set.

Dokumentation der Elementfunktionen

String model.type.NamedVariableType.getErrorMessage ()[virtual]

Return the field errorMessage. No side effects.

Rückgabe:

this.errorMessage

Implementiert model.type.AbstractVariableType.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

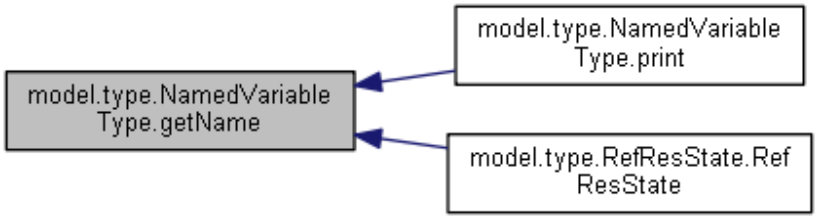
String model.type.NamedVariableType.getName ()

Return the field name. No side effects.

Rückgabe:

this.name

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



Position model.type.NamedVariableType.getPosition ()

Returns the field row. No side effects.

Rückgabe:

this.row

String model.type.NamedVariableType.getRepString ()[virtual]

Returns the type-representation for printing.

Rückgabe:

Typerepresentation for printing.

Implementiert model.type.AbstractVariableType.

String model.type.NamedVariableType.print ()[virtual]

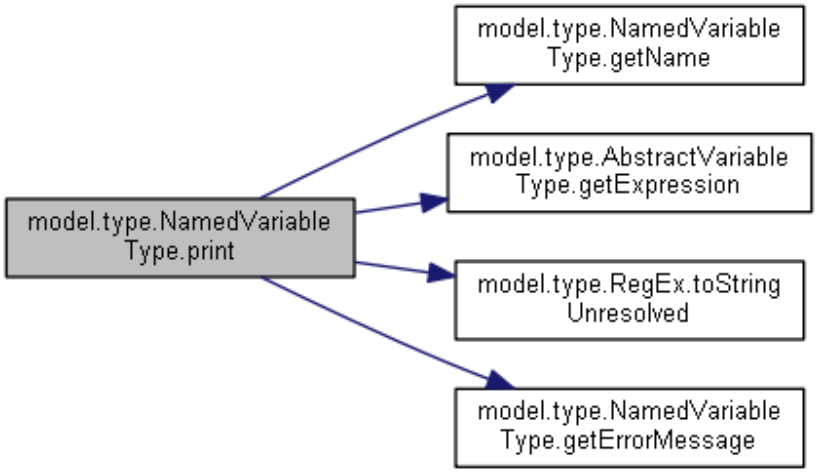
Returns the line to print.

Rückgabe:

Returns a string value, representing a line to print.

Implementiert model.type.AbstractVariableType.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

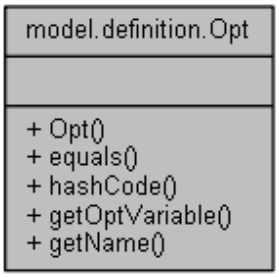


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/type/NamedVariableType.java

model.definition.Opt Klassenreferenz

model.definition.OptZusammengehörigkeiten von model.definition.Opt:



Öffentliche Methoden

- Opt (final String name, final AbstractVariableDefinition optVariable)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- AbstractVariableDefinition getOptVariable ()
- String getName ()

Ausführliche Beschreibung

Representation for an Opt.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.definition.Opt.Opt (final String *name*, final AbstractVariableDefinition *optVariable*)

Constructor, just sets fields. No side effects.

Parameter:

name	The name to set.
optVariable	The opt definition to set.

Dokumentation der Elementfunktionen

String model.definition.Opt.getName ()

Returns the field name. no side effects.

Rückgabe:

this.name

AbstractVariableDefinition model.definition.Opt.getOptVariable ()

Returns the field optVariable. no side effects.

Rückgabe:

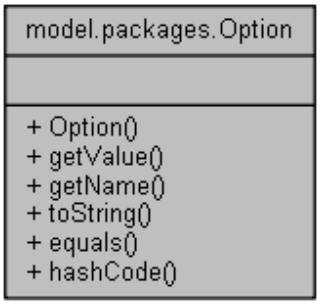
this.optVariable

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/definition/Opt.java

model.packages.Option Klassenreferenz

model.packages.OptionZusammengehörigkeiten von model.packages.Option:



Öffentliche Methoden

- Option (final String name, final String value)
- String getValue ()
- String getName ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()

Ausführliche Beschreibung

This class represents the options of the package rule.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

model.packages.Option.Option (final String *name*, final String *value*)

Parameter:

value	of the option
name	of the option

Dokumentation der Elementfunktionen

String model.packages.Option.getName ()

Rückgabe:

the name

String model.packages.Option.getValue ()

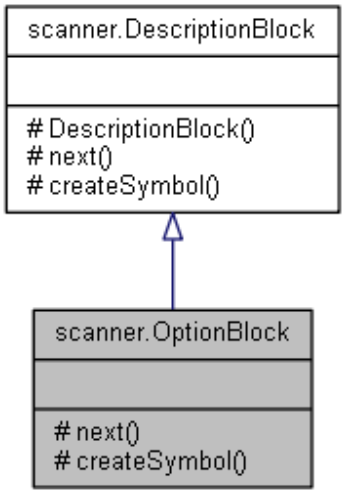
Rückgabe:
the value

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

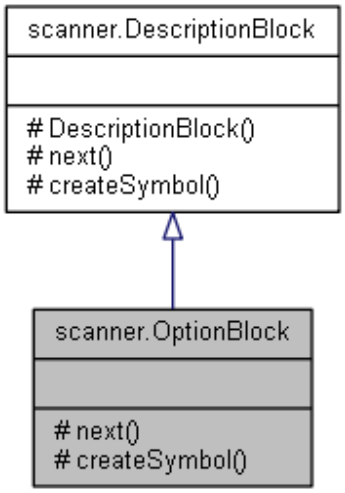
- workspace/ipfli4l13/src/model/packages/Option.java

scanner.OptionBlock Klassenreferenz

scanner.OptionBlockKlassendiagramm für scanner.OptionBlock:



Zusammengehörigkeiten von scanner.OptionBlock:



Geschützte Methoden

- scanner.DescriptionBlock next ()
- AbstractSymbol createSymbol (final String collectedContent, final Position position)

Ausführliche Beschreibung

This class represents the >4 block. The options.

Autor:
Hendrik

Dokumentation der Elementfunktionen

AbstractSymbol scanner.OptionBlock.createSymbol (final String *collectedContent*, final Position *position*) [protected], [virtual]

Create the new Symbol.

Parameter:

collectedContent	collected Content which is used for the Symbol.
position	the position of the collected content.

Rückgabe:
the new Symbol

Implementiert scanner.DescriptionBlock.

scanner.DescriptionBlock scanner.OptionBlock.next () [protected], [virtual]

set the next State.

Rückgabe:
the new State

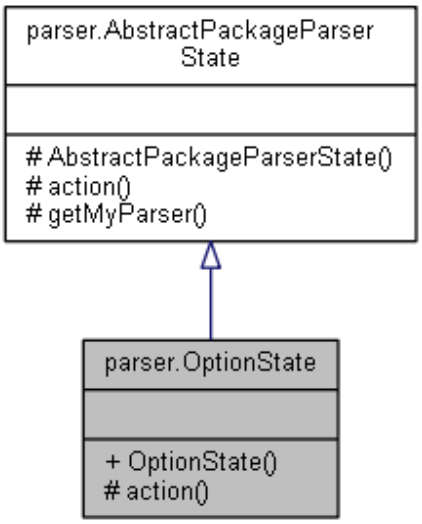
Implementiert scanner.DescriptionBlock.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

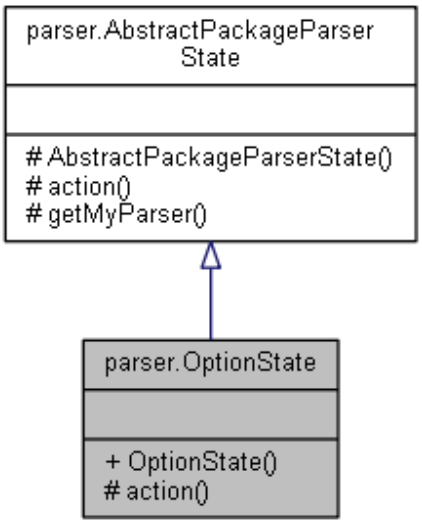
- workspace/ipfli4l13/src/scanner/OptionBlock.java

parser.OptionState Klassenreferenz

parser.OptionStateKlassendiagramm für parser.OptionState:



Zusammengehörigkeiten von parser.OptionState:



Öffentliche Methoden

- OptionState (final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule)

Geschützte Methoden

- AbstractPackageParserState **action** (final PackageDescriptionFileSymbol symbol, final Map< String, AbstractVariableDefinition > variableDef) throws InterruptedException, ParserException

Ausführliche Beschreibung

This class represents the OptionState for the PackageDescriptionfileParser.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.OptionState.OptionState (final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*)

The constructor.

Parameter:

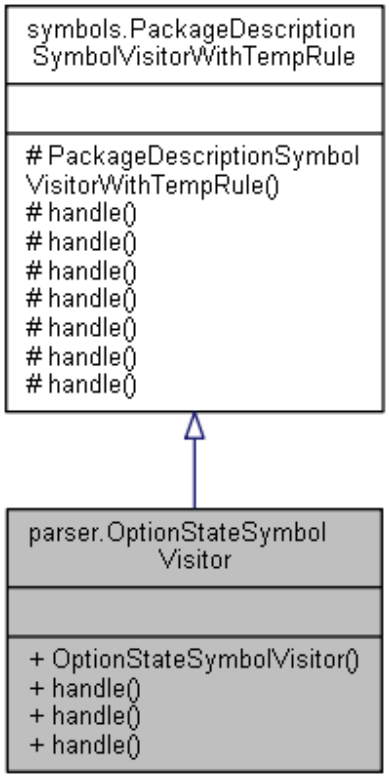
parser	the DescriptionfileParser.
tempPackageRule	The temp package rule.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

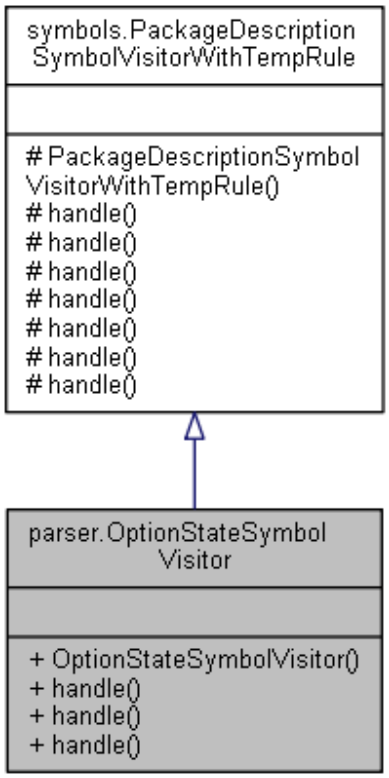
- workspace/ipfli4l13/src/parser/OptionState.java

parser.OptionStateSymbolVisitor Klassenreferenz

parser.OptionStateSymbolVisitorKlassendiagramm für parser.OptionStateSymbolVisitor:



Zusammengehörigkeiten von parser.OptionStateSymbolVisitor:



Öffentliche Methoden

- OptionStateSymbolVisitor ()
- AbstractPackageParserState **handle** (final CommentSymbol commentSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule)
- AbstractPackageParserState **handle** (final RowEndSymbol rowEndSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException
- AbstractPackageParserState **handle** (final OptionSymbol optionSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represent the Visitor for the OptionstateSymbol.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.OptionStateSymbolVisitor.OptionStateSymbolVisitor ()

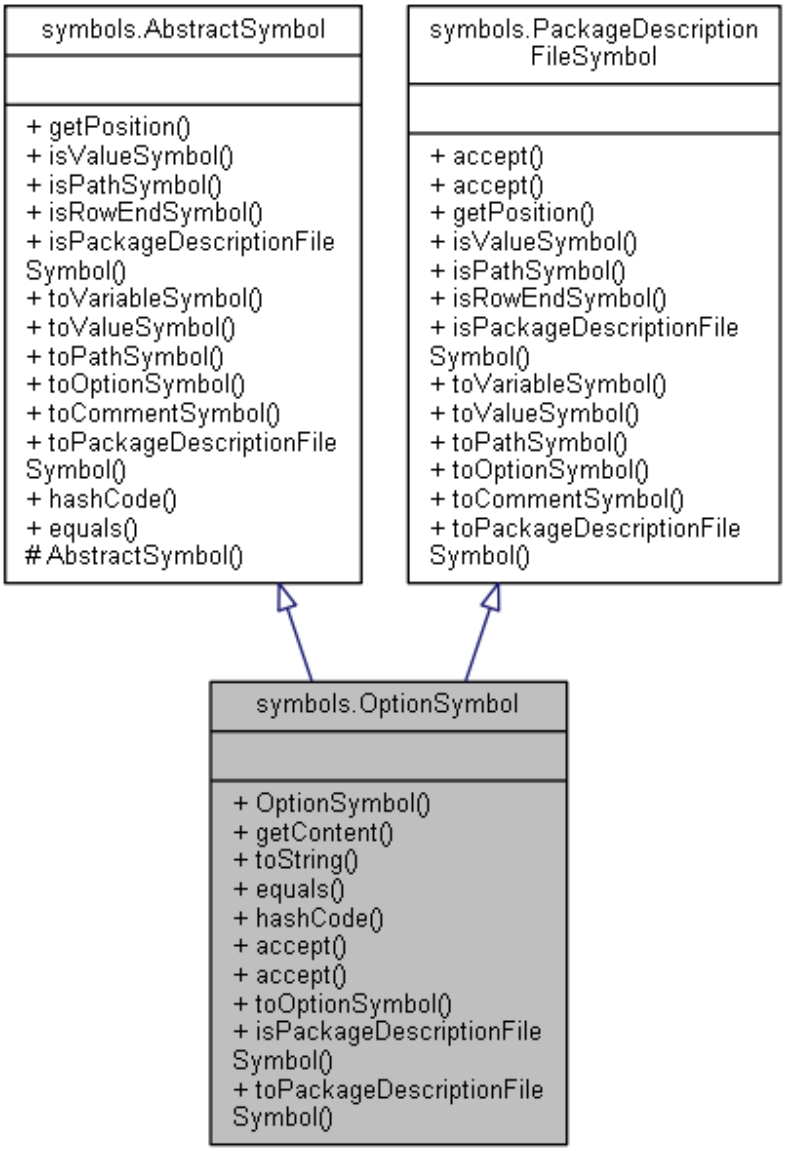
constructor.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

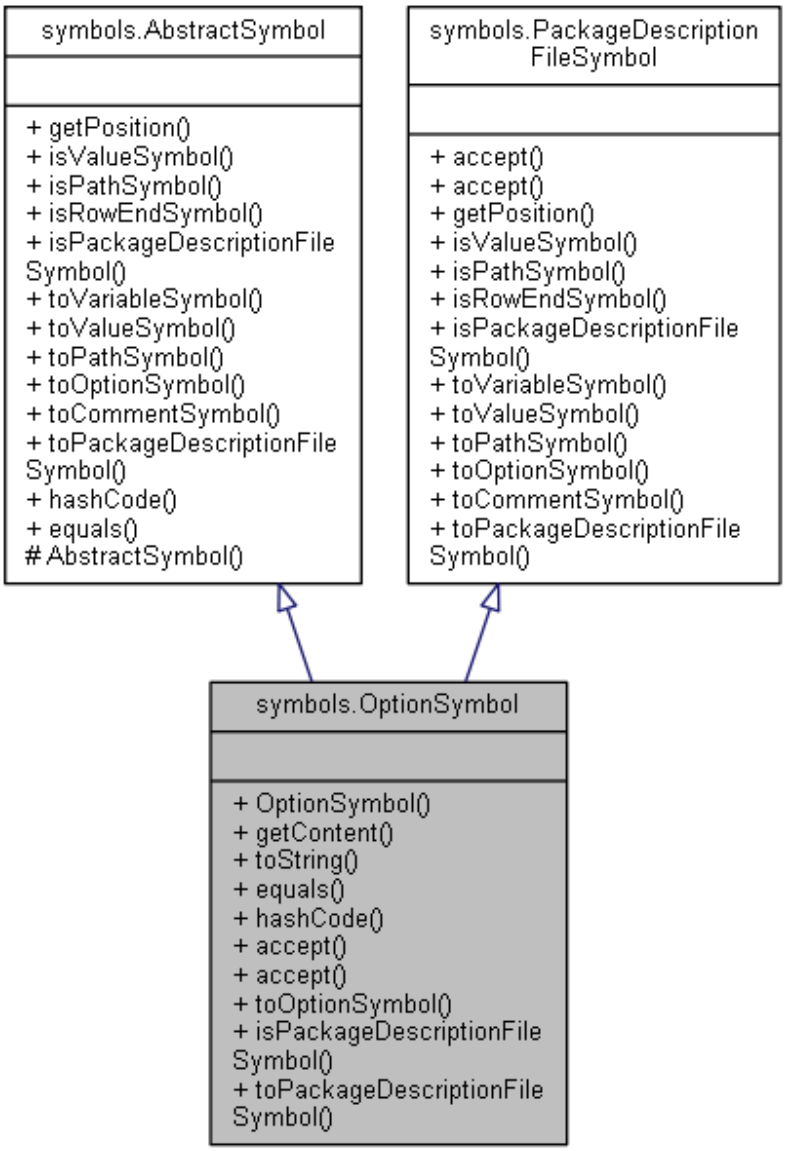
- workspace/ipfli4l13/src/parser/OptionStateSymbolVisitor.java

symbols.OptionSymbol Klassenreferenz

symbols.OptionSymbolKlassendiagramm für symbols.OptionSymbol:



Zusammengehörigkeiten von symbols.OptionSymbol:



Öffentliche Methoden

- OptionSymbol (final String content, final Position position)
- String getContent ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitor visitor, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitorWithTempRule visitor, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- OptionSymbol toOptionSymbol ()
- Boolean isPackageDescriptionFileSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the option symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.OptionSymbol.OptionSymbol (final String *content*, final Position *position*)

This constructor instantiate the option symbol.

Parameter:

position	is the position of the data in the stream.
content	is the content of the OptionState.

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.OptionSymbol.accept (final PackageDescriptionSymbolVisitor *visitor*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

AbstractPackageParserState symbols.OptionSymbol.accept (final PackageDescriptionSymbolVisitorWithTempRule *visitor*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackage-Rule*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.
tempPackageRule	is a temporary package rule.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

String symbols.OptionSymbol.getContent ()

Rückgabe:

the content of the OptionSymbol.

Boolean symbols.OptionSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

OptionSymbol symbols.OptionSymbol.toOptionSymbol ()

Rückgabe:

an error, if the symbol is not an option symbol

Implementiert symbols.PackageDescriptionFileSymbol.

PackageDescriptionFileSymbol symbols.OptionSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:

a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.

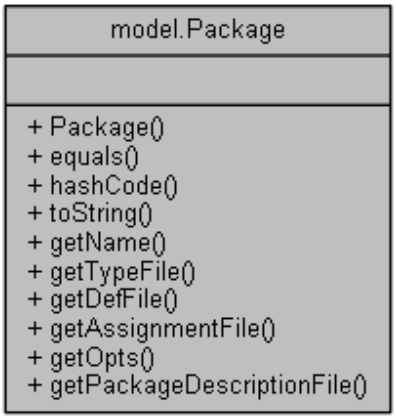
Implementiert symbols.PackageDescriptionFileSymbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/symbols/OptionSymbol.java

model.Package Klassenreferenz

model.PackageZusammengehörigkeiten von model.Package:



Öffentliche Methoden

- Package (final String name, final TypeFile typeFile, final DefinitionFile defFile, final AssignmentFile assignmentFile, final Map< String, Opt > opts2, final PackageDescriptionFile packageDescFile)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- String **toString** ()
- String getName ()
- TypeFile getTypeFile ()
- DefinitionFile getDefFile ()
- AssignmentFile getAssignmentFile ()
- Map< String, Opt > getOpts ()
- PackageDescriptionFile getPackageDescriptionFile ()

Ausführliche Beschreibung

Represents a Package, for example the base package of fli4l.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.Package.Package (final String *name*, final TypeFile *typeFile*, final DefinitionFile *defFile*, final AssignmentFile *assignmentFile*, final Map< String, Opt > *opts2*, final PackageDescriptionFile *packageDescFile*)

Constructor, just sets fields. No side effect.

Parameter:

name	The name to set

typeFile	the type file to set
defFile	the definition file to set
assignmentFile	the assignment file to set
opts2	the opts to set
packageDescFile	to set the packageDescriptionFile

Dokumentation der Elementfunktionen

AssignmentFile model.Package.getAssignmentFile ()

Returns the field assignmentFile. No side effects.

Rückgabe:

this.assignmentFile

DefinitionFile model.Package.getDefFile ()

Returns the field defFile. No side effects.

Rückgabe:

this.defFile

String model.Package.getName ()

Returns the field name. No side effects.

Rückgabe:

this.name

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

Map<String, Opt> model.Package.getOpts ()

Returns the field opts. No side effects.

Rückgabe:

this.opts

PackageDescriptionFile model.Package.getPackageDescriptionFile ()

Rückgabe:

the packageDescriptionFile

TypeFile model.Package.getTypeFile ()

Returns the field typeFile. No side effects.

Rückgabe:

this.typeFile

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/Package.java

model.packages.PackageDescriptionFile Klassenreferenz

model.packages.PackageDescriptionFileZusammengehörigkeiten von model.packages.PackageDescriptionFile:

Öffentliche Methoden

- PackageDescriptionFile (final List< Comment > comments, final List< PackageRule > packagerules, final Integer version)
- String **toString** ()
- Integer getFormatVersion ()
- List< Comment > getComments ()
- List< PackageRule > getPackagerules ()
- int **hashCode** ()
- boolean **equals** (final Object obj)

Ausführliche Beschreibung

This class represents the description files for the packages.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

model.packages.PackageDescriptionFile.PackageDescriptionFile (final List< Comment > *comments*, final List< PackageRule > *packagerules*, final Integer *version*)

Parameter:

comments	represents a list of comments.
packagerules	represents the list of package rules.
version	is the format version of the PackageDescriptionFile

Dokumentation der Elementfunktionen

List<Comment> model.packages.PackageDescriptionFile.getComments ()

Rückgabe:
the comments

Integer model.packages.PackageDescriptionFile.getFormatVersion ()

Rückgabe:
the formatVersion

List<PackageRule> model.packages.PackageDescriptionFile.getPackagerules ()

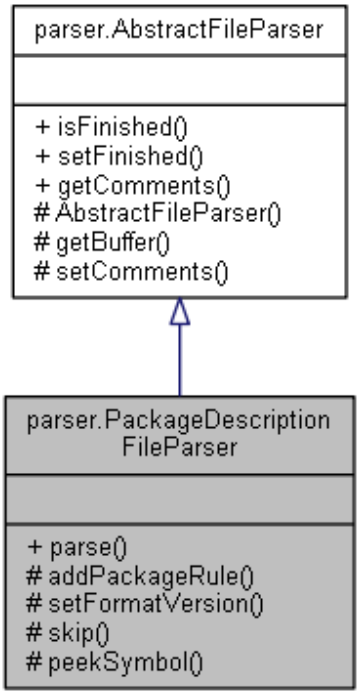
Rückgabe:
the packagerules

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

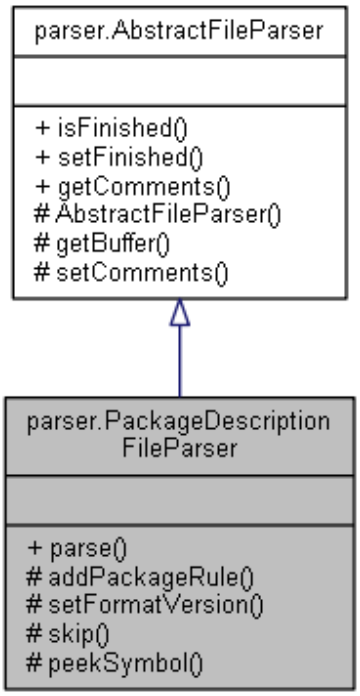
- workspace/ipfli4l13/src/model/packages/PackageDescriptionFile.java

parser.PackageDescriptionFileParser Klassenreferenz

parser.PackageDescriptionFileParserKlassendiagramm für parser.PackageDescriptionFileParser:



Zusammengehörigkeiten von parser.PackageDescriptionFileParser:



Öffentliche, statische Methoden

- static PackageDescriptionFile parse (final Buffer< AbstractSymbol > buffer, final Map< String, AbstractVariableDefinition > variableDef) throws InterruptedException, ParserException

Geschützte Methoden

- void addPackageRule (final PackageRule packageRule)
- void setFormatVersion (final Integer version)
- void skip () throws InterruptedException
- PackageDescriptionFileSymbol peekSymbol () throws InterruptedException, ParserException

Weitere Geerbte Elemente

Ausführliche Beschreibung

Parser for the PackageDescription File.

Autor:

HFW410

Dokumentation der Elementfunktionen

void parser.PackageDescriptionFileParser.addPackageRule (final PackageRule *packageRule*)[protected]

adds a new Packagerule to the List of the PackageRules.

Parameter:

packageRule	new one

static PackageDescriptionFile parser.PackageDescriptionFileParser.parse (final Buffer< AbstractSymbol > *buffer*, final Map< String, AbstractVariableDefinition > *variableDef*) throws InterruptedException, ParserException [static]

creates a PackageDescriptionFileParser and starts the parse-Method.

Parameter:

buffer	is the buffer with the symbols.
variableDef	is the variableDefinitions of the Assignmentfileparser

Rückgabe:

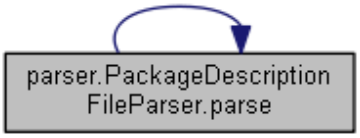
a PackageDescriptionFile

Ausnahmebehandlung:

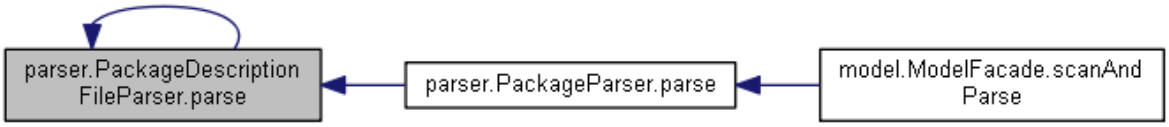
InterruptedEx-	if there is any problem with the buffer.

ception	
ParserException	if there is any problem during the parse-methode.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



PackageDescriptionFileSymbol parser.PackageDescriptionFileParser.peekSymbol () throws InterruptedException, ParserException [protected]

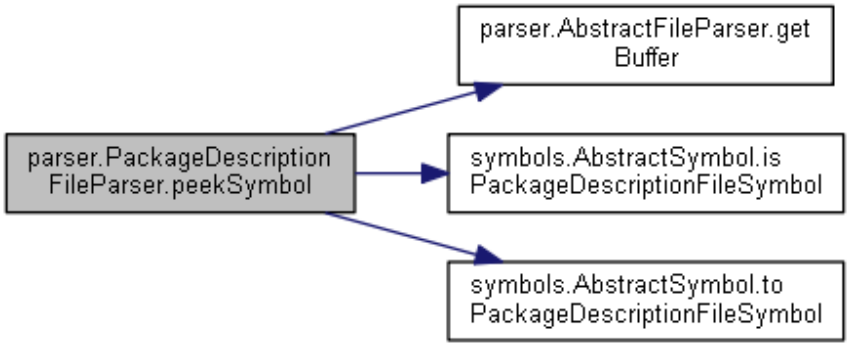
Rückgabe:

a copy of the next symbol in the buffer.

Ausnahmebehandlung:

InterruptedException	from the buffer.
ParserException	if the next Symbol is no package description file symbol.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



void parser.PackageDescriptionFileParser.setFormatVersion (final Integer *version*)[protected]

sets the format version.

Parameter:

version	is the format version of the package description.

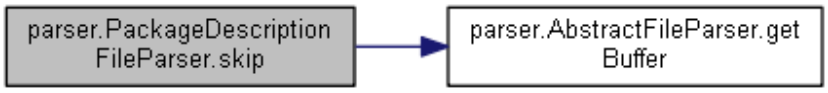
void parser.PackageDescriptionFileParser.skip () throws InterruptedException [protected]

removes the first symbol of the buffer.

Ausnahmebehandlung:

InterruptedException	from the buffer.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

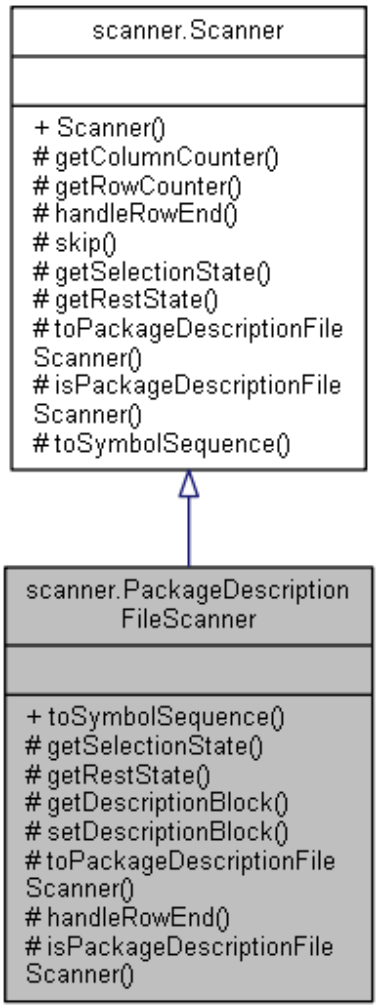


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

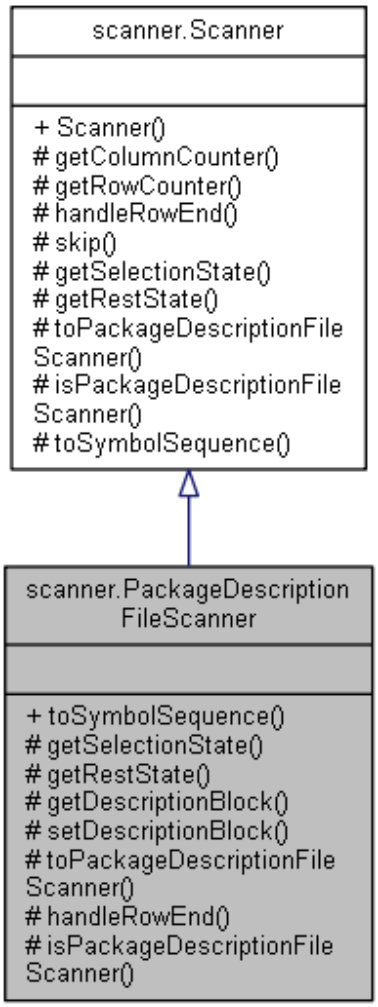
- workspace/ipfli4l13/src/parser/PackageDescriptionFileParser.java

scanner.PackageDescriptionFileScanner Klassenreferenz

scanner.PackageDescriptionFileScannerKlassendiagramm für scanner.PackageDescriptionFileScanner:



Zusammengehörigkeiten von scanner.PackageDescriptionFileScanner:



Öffentliche, statische Methoden

- static void toSymbolSequence (final InputStream input, final String dataPath, final Buffer< AbstractSymbol > buffer) throws ScannerException, InterruptedException, IOException

Geschützte Methoden

- AbstractState getSelectionState ()
- AbstractState **getRestState** (final Integer beginningRow, final Integer beginningColumn)
- DescriptionBlock getDescriptionBlock ()
- void setDescriptionBlock (final DescriptionBlock descriptionBlock)
- PackageDescriptionFileScanner toPackageDescriptionFileScanner ()
- void **handleRowEnd** ()
- Boolean **isPackageDescriptionFileScanner** ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

scanner for package description files.

Autor:

Lisa Leitloff

Dokumentation der Elementfunktionen

DescriptionBlock scanner.PackageDescriptionFileScanner.getDescriptionBlock ()[protected]

Rückgabe:

the descriptionBlock

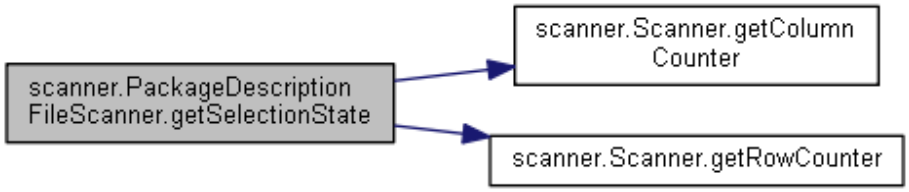
AbstractState scanner.PackageDescriptionFileScanner.getSelectionState ()[protected], [virtual]

Rückgabe:

the SelectionState of the special scanner.

Implementiert scanner.Scanner.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



void scanner.PackageDescriptionFileScanner.setDescriptionBlock (final DescriptionBlock *descriptionBlock*)[protected]

Parameter:

descriptionBlock	the descriptionBlock to set

PackageDescriptionFileScanner scanner.PackageDescriptionFileScanner.toPackageDescriptionFileScanner ()[protected]

Rückgabe:

the PackageDescriptionFileScanner.

static void scanner.PackageDescriptionFileScanner.toSymbolSequence (final InputStream *input*, final String *dataPath*, final Buffer< AbstractSymbol > *buffer*) throws ScannerException, InterruptedException, IOException `[static]`

creates a scanner and starts him.

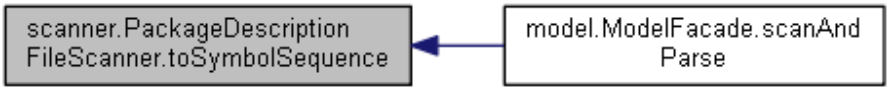
Parameter:

input	is the input stream.
dataPath	is the path of the file.
buffer	if the buffer for the symbols.

Ausnahmebehandlung:

IOException	from the Input Stream.
InterruptedException	from the buffer.
ScannerException	if there is any problem during the scan.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

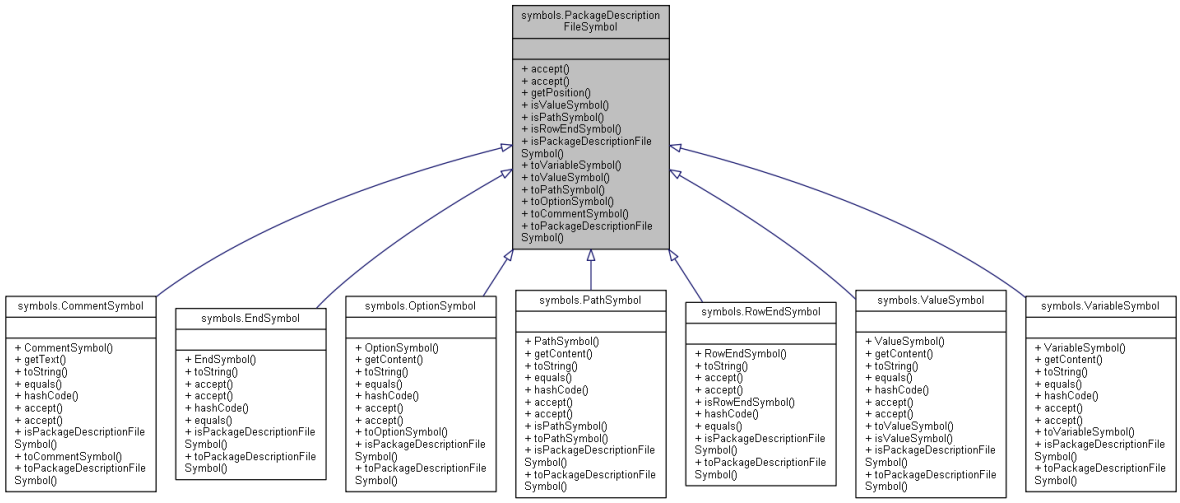


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

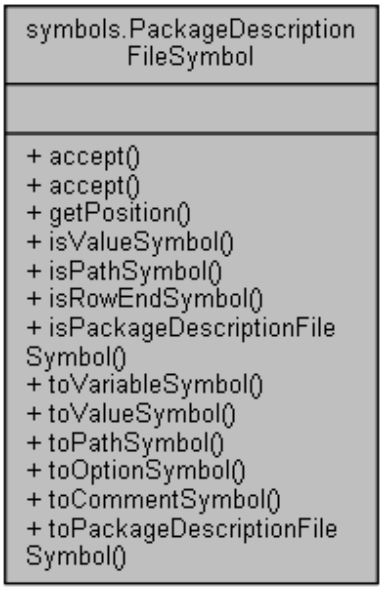
- workspace/ipfli4l13/src/scanner/PackageDescriptionFileScanner.java

symbols.PackageDescriptionFileSymbol Schnittstellenreferenz

symbols.PackageDescriptionFileSymbolKlassendiagramm für symbols.PackageDescriptionFileSymbol:



Zusammengehörigkeiten von symbols.PackageDescriptionFileSymbol:



Öffentliche Methoden

- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitor visitor, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException

- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitorWithTempRule visitor, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- Position getPosition ()
- Boolean isValueSymbol ()
- Boolean isPathSymbol ()
- Boolean isRowEndSymbol ()
- Boolean isPackageDescriptionFileSymbol ()
- VariableSymbol toVariableSymbol ()
- ValueSymbol toValueSymbol ()
- PathSymbol toPathSymbol ()
- OptionSymbol toOptionSymbol ()
- CommentSymbol toCommentSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()

Ausführliche Beschreibung

represents a PackageDescriptionFileSymbol.

Autor:

Lisa Leitloff

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.PackageDescriptionFileSymbol.accept (final PackageDescriptionSymbolVisitor *visitor*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert in symbols.CommentSymbol, symbols.OptionSymbol, symbols.PathSymbol, symbols.ValueSymbol, symbols.VariableSymbol, symbols.EndSymbol und symbols.RowEndSymbol.

AbstractPackageParserState symbols.PackageDescriptionFileSymbol.accept (final PackageDescriptionSymbolVisitorWithTempRule *visitor*, final

PackageDescriptionFileParser ***parser*****, final TempPackageRule** ***tempPackage-Rule*****) throws InterruptedException, ParserException**

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.
tempPackageRule	is a temporary package rule.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert in symbols.CommentSymbol, symbols.OptionSymbol, symbols.PathSymbol, symbols.ValueSymbol, symbols.VariableSymbol, symbols.EndSymbol und symbols.RowEndSymbol.

Position symbols.PackageDescriptionFileSymbol.getPosition ()

Rückgabe:

the position

Boolean sym-bols.PackageDescriptionFileSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Implementiert in symbols.PathSymbol, symbols.ValueSymbol, symbols.OptionSymbol, symbols.VariableSymbol, symbols.CommentSymbol, symbols.RowEndSymbol und symbols.EndSymbol.

Boolean symbols.PackageDescriptionFileSymbol.isPathSymbol ()

Rückgabe:

true, if the Symbol is a path symbol.

Implementiert in symbols.PathSymbol.

Boolean symbols.PackageDescriptionFileSymbol.isRowEndSymbol ()

Rückgabe:

true, if the Symbol is a row end symbol.

Implementiert in symbols.RowEndSymbol.

Boolean symbols.PackageDescriptionFileSymbol.isValueSymbol ()

Rückgabe:
true, if the Symbol is a value symbol.
Implementiert in symbols.ValueSymbol.

CommentSymbol symbols.PackageDescriptionFileSymbol.toCommentSymbol ()

Rückgabe:
an error, if the symbol is not a comment symbol
Implementiert in symbols.CommentSymbol.

OptionSymbol symbols.PackageDescriptionFileSymbol.toOptionSymbol ()

Rückgabe:
an error, if the symbol is not an option symbol
Implementiert in symbols.OptionSymbol.

PackageDescriptionFileSymbol symbols.PackageDescriptionFileSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:
a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.
Implementiert in symbols.PathSymbol, symbols.ValueSymbol, symbols.CommentSymbol, symbols.OptionSymbol, symbols.VariableSymbol, symbols.RowEndSymbol und symbols.EndSymbol.

PathSymbol symbols.PackageDescriptionFileSymbol.toPathSymbol ()

Rückgabe:
an error, if the symbol is not a path symbol
Implementiert in symbols.PathSymbol.

ValueSymbol symbols.PackageDescriptionFileSymbol.toValueSymbol ()

Rückgabe:
an error, if the symbol is not a value symbol
Implementiert in symbols.ValueSymbol.

VariableSymbol symbols.PackageDescriptionFileSymbol.toVariableSymbol ()

Rückgabe:
an error, if the symbol is not a variable symbol

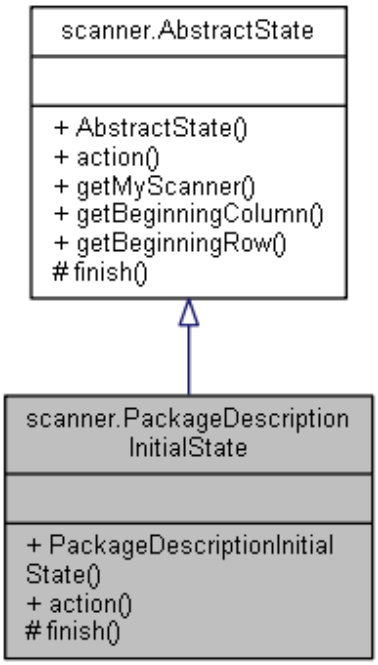
Implementiert in symbols.VariableSymbol.

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

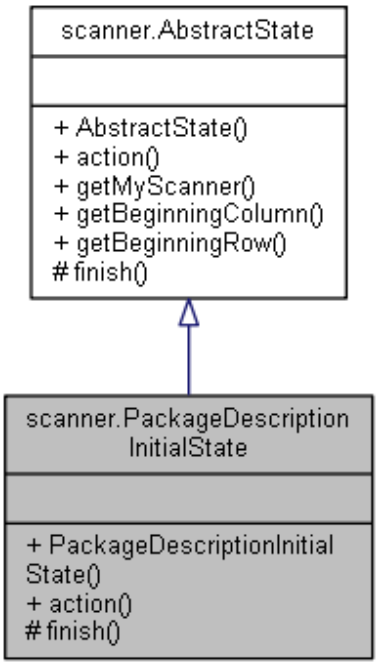
- workspace/ipfli4l13/src/symbols/PackageDescriptionFileSymbol.java

scanner.PackageDescriptionInitialState Klassenreferenz

scanner.PackageDescriptionInitialStateKlassendiagramm für scanner.PackageDescriptionInitialState:



Zusammengehörigkeiten von scanner.PackageDescriptionInitialState:



Öffentliche Methoden

- PackageDescriptionInitialState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)
- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath)

Ausführliche Beschreibung

The Selectionstate for the PackageDefinitionFileScanner.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

scanner.PackageDescriptionInitialState.PackageDescriptionInitialState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

creates a PackageDefinitionSelectionState.

Parameter:

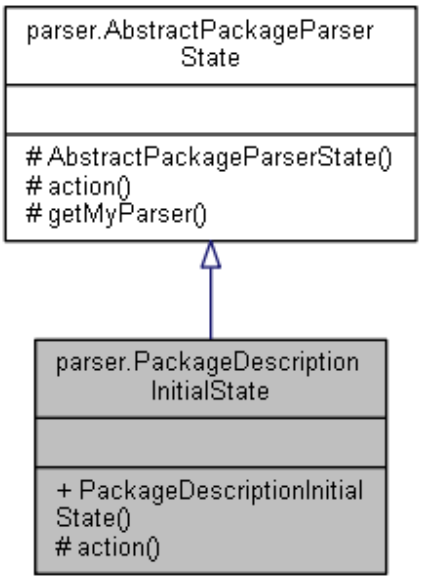
myScanner	is the scanner with owns this state.
beginningColumn	is the first column of the Symbol.
beginningRow	is the first row of the Symbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

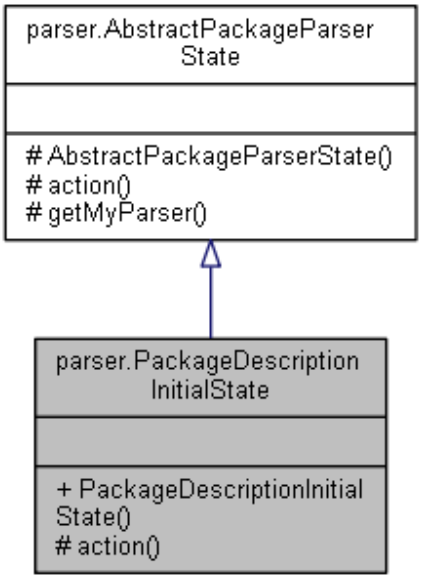
- workspace/ipfli4l13/src/scanner/PackageDescriptionInitialState.java

parser.PackageDescriptionInitialState Klassenreferenz

parser.PackageDescriptionInitialStateKlassendiagramm für parser.PackageDescriptionInitialState:



Zusammengehörigkeiten von parser.PackageDescriptionInitialState:



Öffentliche Methoden

- PackageDescriptionInitialState (final PackageDescriptionFileParser parser)

Geschützte Methoden

- AbstractPackageParserState **action** (final PackageDescriptionFileSymbol symbol, final Map< String, AbstractVariableDefinition > variableDef) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

The InititalState for the PackageDescriptionParser.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.PackageDescriptionInitialState.PackageDescriptionInitialState (final PackageDescriptionFileParser *parser*)

creates a PackageDescriptionInitialState.

Parameter:

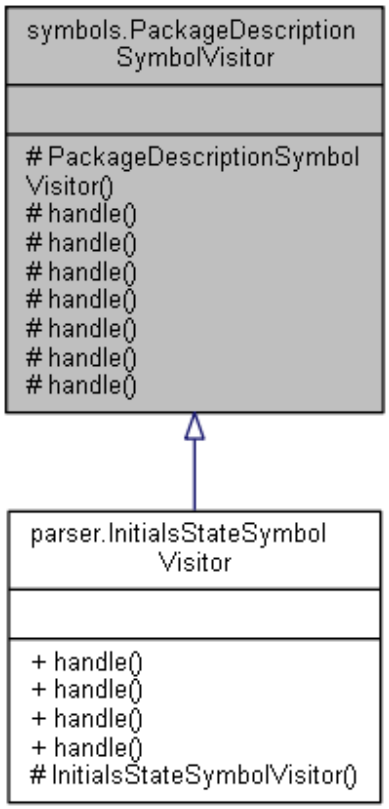
parser	is the parser, who owns the state.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

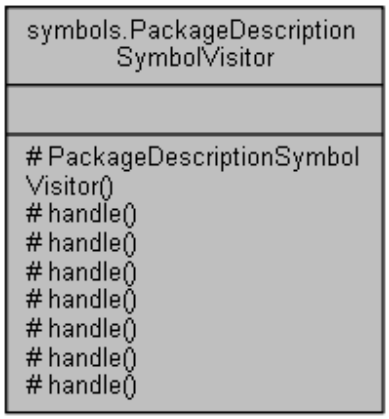
- workspace/ipfli4l13/src/parser/PackageDescriptionInitialState.java

symbols.PackageDescriptionSymbolVisitor Klassenreferenz

symbols.PackageDescriptionSymbolVisitorKlassendiagramm für symbols.PackageDescriptionSymbolVisitor:



Zusammengehörigkeiten von symbols.PackageDescriptionSymbolVisitor:



Geschützte Methoden

- PackageDescriptionSymbolVisitor (final String exceptionMessage)
- AbstractPackageParserState handle (final CommentSymbol commentSymbol, final PackageDescriptionFileParser parser) throws ParserException
- AbstractPackageParserState handle (final EndSymbol endSymbol, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState handle (final RowEndSymbol rowEndSymbol, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState handle (final VariableSymbol variableSymbol, final PackageDescriptionFileParser parser) throws ParserException
- AbstractPackageParserState handle (final ValueSymbol valueSymbol, final PackageDescriptionFileParser parser) throws ParserException
- AbstractPackageParserState handle (final PathSymbol pathSymbol, final PackageDescriptionFileParser parser) throws ParserException
- AbstractPackageParserState handle (final OptionSymbol optionSymbol, final PackageDescriptionFileParser parser) throws ParserException

---

Ausführliche Beschreibung

Visitor for the Symbols.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

symbols.PackageDescriptionSymbolVisitor.PackageDescriptionSymbolVisitor (final String *exceptionMessage*)`[protected]`

konstruktor of the SymbolVisitor.

Parameter:

exceptionMessage	is the message for the parserexception, which is thrown

---

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitor.handle (final CommentSymbol *commentSymbol*, final PackageDescriptionFileParser *parser*) throws ParserException `[protected]`

handles an object of the type commentSymbol.

Parameter:

commentSymbol	is the type of the handled object.
parser	is the parser.

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitor.handle (final EndSymbol *endSymbol*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException [protected]

handles an object of the Type EndSymbol.

Parameter:

endSymbol	is the type of the handled object.
parser	is the parser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitor.handle (final RowEndSymbol *rowEndSymbol*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException [protected]

handles an object of the Type RowEndSymbol.

Parameter:

rowEndSymbol	is the type of the handled object.
parser	is the parser.

Ausnahmebehandlung:

InterruptedException	from the buffer.

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitor.handle (final VariableSymbol *variableSymbol*, final PackageDescriptionFileParser *parser*) throws ParserException [protected]

handles an object of the Type VariableSymbol.

Parameter:

variableSymbol	is the type of the handled object.
parser	is the parser.

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitor.handle (final ValueSymbol *valueSymbol*, final PackageDescriptionFileParser *parser*) throws ParserException [protected]

handles an object of the Type ValueSymbol.

Parameter:

valueSymbol	is the type of the handled object.
parser	is the parser.

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitor.handle (final PathSymbol *pathSymbol*, final PackageDescriptionFileParser *parser*) throws ParserException [protected]

handles an object of the Type PathSymbol.

Parameter:

pathSymbol	is the type of the handled object.
parser	is the parser.

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitor.handle (final OptionSymbol *optionSymbol*, final PackageDescriptionFileParser *parser*) throws ParserException `[protected]`

handles an object of the Type OptionSymbol.

Parameter:

optionSymbol	is the type of the handled object.
parser	is the parser.

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

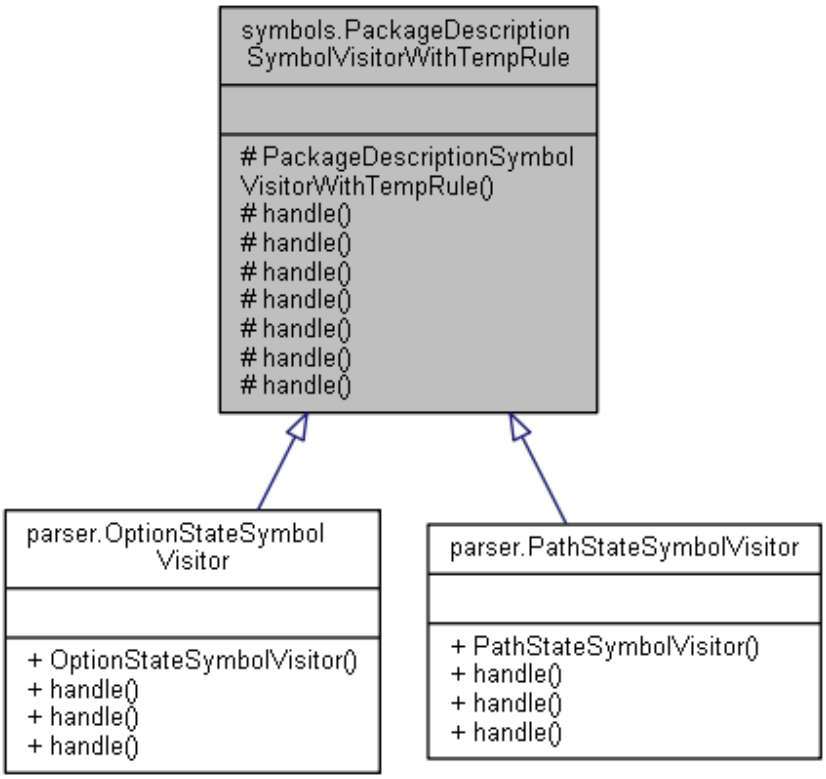
ParserException	if the symbol is not expected.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

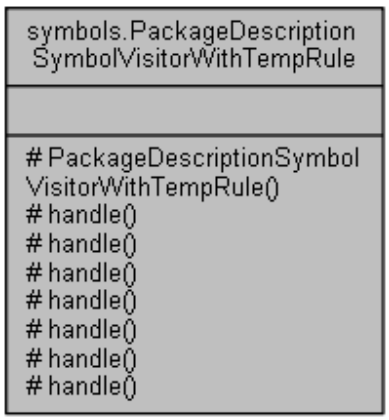
- workspace/ipfli4l13/src/symbols/PackageDescriptionSymbolVisitor.java

symbols.PackageDescriptionSymbolVisitorWithTempRule Klassenreferenz

symbols.PackageDescriptionSymbolVisitorWithTempRuleKlassendiagramm für symbols.PackageDescriptionSymbolVisitorWithTempRule:



Zusammengehörigkeiten von symbols.PackageDescriptionSymbolVisitorWithTempRule:



Geschützte Methoden

- PackageDescriptionSymbolVisitorWithTempRule (final String exceptionMessage)

- AbstractPackageParserState handle (final CommentSymbol commentSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws ParserException
- AbstractPackageParserState handle (final EndSymbol endSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- AbstractPackageParserState handle (final RowEndSymbol rowEndSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- AbstractPackageParserState handle (final VariableSymbol variableSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws ParserException
- AbstractPackageParserState handle (final ValueSymbol valueSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws ParserException
- AbstractPackageParserState handle (final PathSymbol pathSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws ParserException
- AbstractPackageParserState handle (final OptionSymbol optionSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws ParserException

---

Ausführliche Beschreibung

Visitor for the Symbols. This visitor transmit a TempPackageRule in the handle-method.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

symbols.PackageDescriptionSymbolVisitorWithTempRule.PackageDescriptionSymbolVisitorWithTempRule (final String *exceptionMessage*)`[protected]`

constructor for the SymbolVisitorWithTempRule.

Parameter:

exceptionMessage	for the ParserExceptuin, which will be thrown.

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitorWithTempRule.handle (final CommentSymbol *commentSymbol*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws ParserException `[protected]`

handles an object of the type commentSymbol.

Parameter:

commentSymbol	is the type of the handled object.

parser	is the parser.
tempPackageRule	is the temporary Package Rule

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitorWithTempRule.handle (final EndSymbol *endSymbol*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws InterruptedException, ParserException [protected]

handles an object of the Type EndSymbol.

Parameter:

endSymbol	is the type of the handled object.
parser	is the parser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.

Parameter:

tempPackageRule	is the temporary Package Rule

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitorWithTempRule.handle (final RowEndSymbol *rowEndSymbol*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws InterruptedException, ParserException [protected]

handles an object of the Type RowEndSymbol.

Parameter:

rowEndSymbol	is the type of the handled object.
parser	is the parser.
tempPackageRule	is the temporary Package Rule

Ausnahmebehandlung:

InterruptedException	from the buffer.

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState sym-bols.PackageDescriptionSymbolVisitorWithTempRule.handle (final VariableSymbol *variableSymbol*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws ParserException `[protected]`

handles an object of the Type VariableSymbol.

Parameter:

variableSymbol	is the type of the handled object.
parser	is the parser.
tempPackageRule	is the temporary Package Rule

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitorWithTempRule.handle (final ValueSymbol *valueSymbol*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws ParserException `[protected]`

handles an object of the Type ValueSymbol.

Parameter:

valueSymbol	is the type of the handled object.
parser	is the parser.
tempPackageRule	is the temporary Package Rule

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitorWithTempRule.handle (final PathSymbol *pathSymbol*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws ParserException `[protected]`

handles an object of the Type PathSymbol.

Parameter:

pathSymbol	is the type of the handled object.
parser	is the parser.
tempPackageRule	is the temporary Package Rule

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

ParserException	if the symbol is not expected.

AbstractPackageParserState symbols.PackageDescriptionSymbolVisitorWithTempRule.handle (final OptionSymbol *optionSymbol*, final PackageDescriptionFileParser *parser*, final

TempPackageRule *tempPackageRule*) throws ParserException [protected]

handles an object of the Type OptionSymbol.

Parameter:

optionSymbol	is the type of the handled object.
parser	is the parser.
tempPackageRule	is the temporary Package Rule

Rückgabe:

the actual state of the parser.

Ausnahmebehandlung:

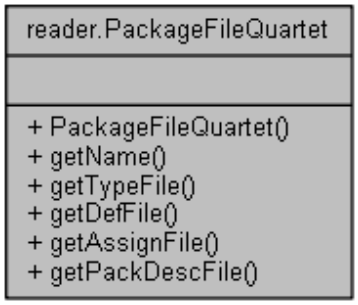
ParserException	if the symbol is not expected.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/symbols/PackageDescriptionSymbolVisitorWithTempRule.java

reader.PackageFileQuartet Klassenreferenz

reader.PackageFileQuartetZusammengehörigkeiten von reader.PackageFileQuartet:



Öffentliche Methoden

- PackageFileQuartet (final String name, final File typeFile, final File defFile, final File assignFile, final File packDescFile) throws ReaderException
- String getName ()
- File getTypeFile ()
- File getDefFile ()
- File getAssignFile ()
- File getPackDescFile ()

Ausführliche Beschreibung

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

reader.PackageFileQuartet.PackageFileQuartet (final String *name*, final File *typeFile*, final File *defFile*, final File *assignFile*, final File *packDescFile*) throws ReaderException

Parameter:

name	The name of the package.
typeFile	The typeFile of the package.
defFile	The defFile of the package.
assignFile	The assignmentFile of the package.

packDescFile	The packagedescriptionfile to set.

Ausnahmebehandlung:

ReaderException	If the reader causes an exception.

Dokumentation der Elementfunktionen

File reader.PackageFileQuartet.getAssignFile ()

Rückgabe:
the assignFile

File reader.PackageFileQuartet.getDefFile ()

Rückgabe:
the defFile

String reader.PackageFileQuartet.getName ()

Rückgabe:
the name

File reader.PackageFileQuartet.getPackDescFile ()

Rückgabe:
the packDescFile

File reader.PackageFileQuartet.getTypeFile ()

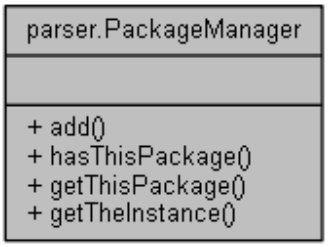
Rückgabe:
the typeFile

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/reader/PackageFileQuartet.java

parser.PackageManager Klassenreferenz

parser.PackageManagerZusammengehörigkeiten von parser.PackageManager:



Öffentliche Methoden

- void add (final Package pkg)
- Boolean hasThisPackage (final String pkgName)
- Package getThisPackage (final String pkgName)

Öffentliche, statische Methoden

- static PackageManager getTheInstance ()

Ausführliche Beschreibung

represents the packageManager.

Autor:

Muri

Dokumentation der Elementfunktionen

void parser.PackageManager.add (final Package *pkg*)

Parameter:

pkg	a package

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

static PackageManager parser.PackageManager.getTheInstance ()[static]

Rückgabe:

the PackageManager.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

Package parser.PackageManager.getThisPackage (final String *pkgName*)

returns the package with this package name.

Parameter:

pkgName	name of the desired Package

Rückgabe:

the package

Boolean parser.PackageManager.hasThisPackage (final String *pkgName*)

returns true, if the package with this Name exists.

Parameter:

pkgName	this Package

Rückgabe:

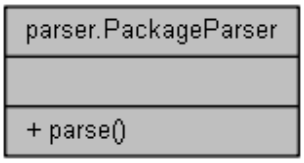
true if the package manager has this package

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/PackageManager.java

parser.PackageParser Klassenreferenz

parser.PackageParserZusammengehörigkeiten von parser.PackageParser:



Öffentliche Methoden

- Package parse (final String pkgName, final Buffer< AbstractSymbol > typeBuffer, final Buffer< AbstractSymbol > definitionBuffer, final Buffer< AbstractSymbol > assignmentBuffer, final Buffer< AbstractSymbol > packDesFileBuffer) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

this is the one and only Package Parser. It generates a Package when calling parse().

Autor:

Muri

Dokumentation der Elementfunktionen

Package parser.PackageParser.parse (final String *pkgName*, final Buffer< AbstractSymbol > *typeBuffer*, final Buffer< AbstractSymbol > *definitionBuffer*, final Buffer< AbstractSymbol > *assignmentBuffer*, final Buffer< AbstractSymbol > *packDesFileBuffer*) throws InterruptedException, ParserException

Parameter:

pkgName	name of the Package
typeBuffer	typeBuffer
definitionBuffer	definitionBuffer
assignmentBuffer	assignmentBuffer
packDesFileBuffer	is a packageDescriptionFileBuffer

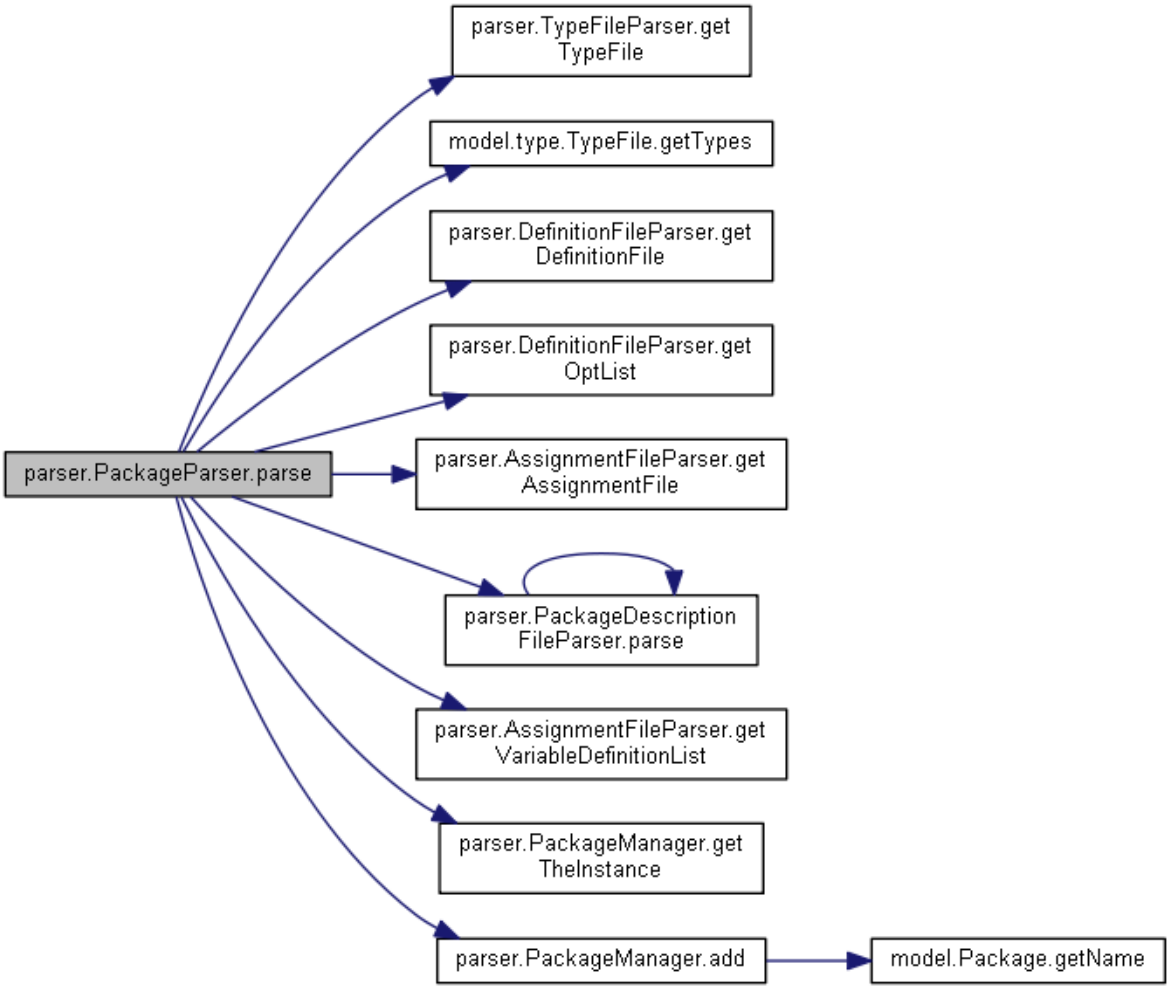
Rückgabe:

the finished package

Ausnahmebehandlung:

ParserException	ParserException
InterruptedException	InterruptedException

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



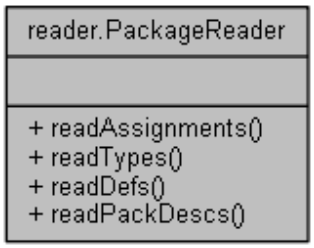
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/PackageParser.java

reader.PackageReader Klassenreferenz

reader.PackageReaderZusammengehörigkeiten von reader.PackageReader:



Öffentliche, statische Methoden

- static File[] readAssignments (final String pathToFli4l)
- static File[] readTypes (final String pathToFli4l)
- static File[] readDefs (final String pathToFli4l)
- static File[] readPackDescs (final String pathToFli4l)

Ausführliche Beschreibung

Autor:
Phil

Dokumentation der Elementfunktionen

static File [] reader.PackageReader.readAssignments (final String *pathToFli4l*)`[static]`

Reads all assignment files.

Parameter:

pathToFli4l	The path to fli4l.

Rückgabe:
parser.PackageManager.

static File [] reader.PackageReader.readDefs (final String *pathToFli4l*)`[static]`

Reads all def files.

Parameter:

pathToFli4l	The path to fli4l.

Rückgabe:
parser.PackageManager.

static File [] reader.PackageReader.readPackDescs (final String *pathToFli4l*)`[static]`

Reads all package description files.

Parameter:

pathToFli4l	The path to fli4l.

Rückgabe:

parser.PackageManager.

static File [] reader.PackageReader.readTypes (final String *pathToFli4l*)`[static]`

Reads all type files.

Parameter:

pathToFli4l	The path to fli4l.

Rückgabe:

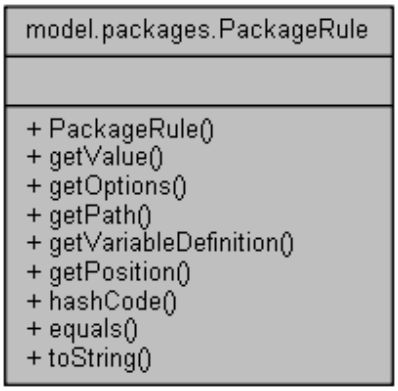
parser.PackageManager.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/reader/PackageReader.java

model.packages.PackageRule Klassenreferenz

model.packages.PackageRuleZusammengehörigkeiten von model.packages.PackageRule:



Öffentliche Methoden

- PackageRule (final RegEx value, final Path path, final List< AbstractVariableDefinition > variableDef, final Position position, final List< Option > options)
- RegEx getValue ()
- List< Option > getOptions ()
- Path getPath ()
- List< AbstractVariableDefinition > getVariableDefinition ()
- Position getPosition ()
- int **hashCode** ()
- boolean **equals** (final Object obj)
- String **toString** ()

Ausführliche Beschreibung

This class represents the rules of the packages in one line of the package description file.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

model.packages.PackageRule.PackageRule (final RegEx *value*, final Path *path*, final List< AbstractVariableDefinition > *variableDef*, final Position *position*, final List< Option > *options*)

Parameter:

value	String of value
options	of the rule

path	of the rule
variableDef	is the list of variableDefinitions, which the assigmentparser owns. rule
position	of the rule

Dokumentation der Elementfunktionen

List<Option> model.packages.PackageRule.getOptions ()

Rückgabe:
the options

Path model.packages.PackageRule.getPath ()

Rückgabe:
the path

Position model.packages.PackageRule.getPosition ()

Rückgabe:
the position

RegEx model.packages.PackageRule.getValue ()

Rückgabe:
the value

List<AbstractVariableDefinition> model.packages.PackageRule.getVariableDefinition ()

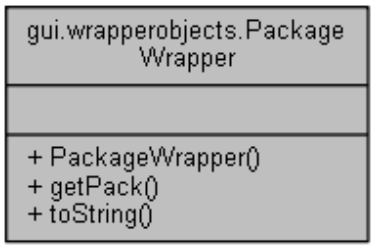
Rückgabe:
the variableDefinition

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/packages/PackageRule.java

gui.wrapperobjects.PackageWrapper Klassenreferenz

gui.wrapperobjects.PackageWrapperZusammengehörigkeiten von gui.wrapperobjects.PackageWrapper:



Öffentliche Methoden

- PackageWrapper (final model.Package pack)
- model.Package getPack ()
- String **toString** ()

Ausführliche Beschreibung

Wrapper-Class for packages.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.wrapperobjects.PackageWrapper.PackageWrapper (final model.Package *pack*)

The packageWrapper constructor.

Parameter:

pack	The pack to wrap.

Dokumentation der Elementfunktionen

model.Package gui.wrapperobjects.PackageWrapper.getPack ()

Rückgabe:

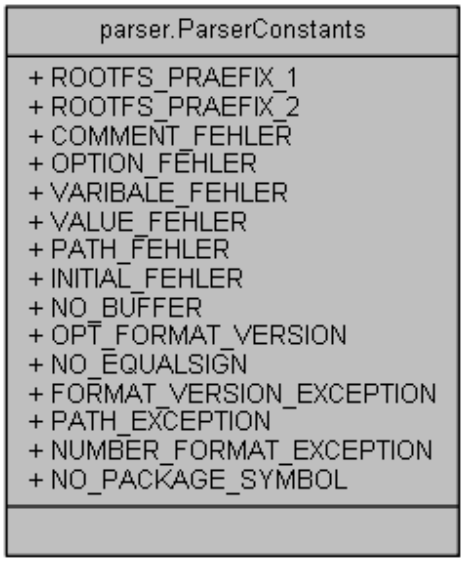
the pack

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/gui/wrapperobjects/PackageWrapper.java

parser.ParserConstants Klassenreferenz

parser.ParserConstantsZusammengehörigkeiten von parser.ParserConstants:



Statische öffentliche Attribute

- static final String ROOTFS_PRAEFIX_1 = "rootfs:"
- static final String ROOTFS_PRAEFIX_2 = "rootfs:files/"
- static final String COMMENT_FEHLER = "after a comment comes no row end."
- static final String OPTION_FEHLER = ""
- static final String VARIBALE_FEHLER = ""
- static final String VALUE_FEHLER = ""
- static final String PATH_FEHLER = ""
- static final String INITIAL_FEHLER = ""
- static final String NO_BUFFER = ""
- static final String OPT_FORMAT_VERSION = "opt_format_version"
- static final String NO_EQUALSIGN = "There is no equalsign in the option"
- static final String FORMAT_VERSION_EXCEPTION
- static final String PATH_EXCEPTION = "The next symbol is no Path Symbol"
- static final String NUMBER_FORMAT_EXCEPTION = "Formatversion has to be an Integer"
- static final String NO_PACKAGE_SYMBOL

Ausführliche Beschreibung

Constrants for the parser.

Autor:

Lisa Leitloff

Dokumentation der Datenelemente

final String parser.ParserConstants.COMMENT_FEHLER = "after a comment comes no row end."[static]

String for the exception in commentstate of PackageDescriptionFileParser.

final String parser.ParserConstants.FORMAT_VERSION_EXCEPTION[static]

```
Initialisierung:=
            "The next symbol is no - as Path Symbol"
```

String for the exception if after a FormatVersion didn't follow a pathSymbol with -.

final String parser.ParserConstants.INITIAL_FEHLER = ""[static]

String for the exception in inital state of PackageDescriptionFileParser.

final String parser.ParserConstants.NO_BUFFER = ""[static]

String for the exception if there is no buffer in PackageDescriptionFileParser.

final String parser.ParserConstants.NO_EQUALSIGN = "There is no equalsign in the option"[static]

String for the exception if there is no equalsign in the option.

final String parser.ParserConstants.NO_PACKAGE_SYMBOL[static]

```
Initialisierung:=
            "The PackageDescriptionFileParser doesn't know this symbol."
```

String for the exception if the packagedescriptionfileparser receive a symbol, he doesn't know.

final String parser.ParserConstants.NUMBER_FORMAT_EXCEPTION = "Formatversion has to be an Integer"[static]

String for the exception if the numberformat excetion will be thrown.

final String parser.ParserConstants.OPT_FORMAT_VERSION = "opt_format_version"[static]

String, which is expected if a format_version exists.

final String parser.ParserConstants.OPTION_FEHLER = ""[static]

String for the exception in OptionState of PackageDescriptionFileParser.

final String parser.ParserConstants.PATH_EXCEPTION = "The next symbol is no Path Symbol"[static]

String for the exception if after a FormatVersion didn't follow a pathSymbol.

final String parser.ParserConstants.PATH_FEHLER = ""[static]

String for the exception in path state of PackageDescriptionFileParser.

final String parser.ParserConstants.ROOTFS_PRAEFIX_1 = "rootfs:"[static]

String praefix for rootfs.

final String parser.ParserConstants.ROOTFS_PRAEFIX_2 = "rootfs:files/"[static]

String praefix for rootfs.

final String parser.ParserConstants.VALUE_FEHLER = ""[static]

String for the exception in Value state of PackageDescriptionFileParser.

final String parser.ParserConstants.VARIBALE_FEHLER = ""[static]

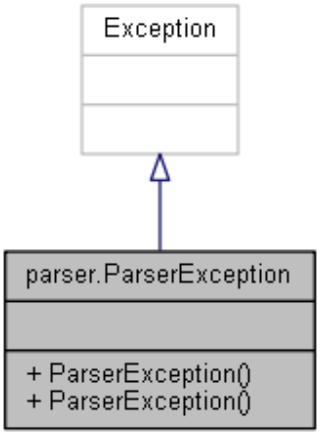
String for the exception in VariableState of PackageDescriptionFileParser.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

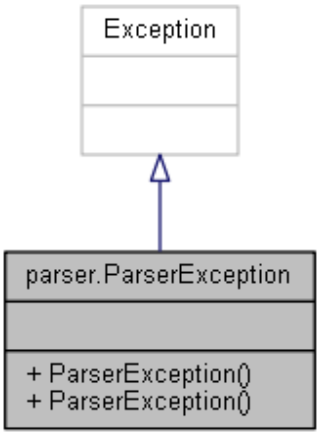
- workspace/ipfli4l13/src/parser/ParserConstants.java

parser.ParserException Klassenreferenz

parser.ParserExceptionKlassendiagramm für parser.ParserException:



Zusammengehörigkeiten von parser.ParserException:



Öffentliche Methoden

- ParserException (final String arg0)
- ParserException (final String message, final Throwable throwable)

Ausführliche Beschreibung

Autor:

Muri

Beschreibung der Konstruktoren und Destruktoren

parser.ParserException.ParserException (final String *arg0*)

Parameter:

arg0	arg0

parser.ParserException.ParserException (final String *message*, final Throwable *throwable*)

Parameter:

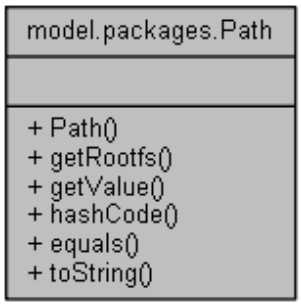
message	is the message for the exception.
throwable	is the throwable which is encapsulated.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/ParserException.java

model.packages.Path Klassenreferenz

model.packages.PathZusammengehörigkeiten von model.packages.Path:



Öffentliche Methoden

- Path (final Boolean rootfs, final String value)
- Boolean getRootfs ()
- String getValue ()
- int **hashCode** ()
- boolean **equals** (final Object obj)
- String **toString** ()

Ausführliche Beschreibung

This class represents the data path.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

model.packages.Path.Path (final Boolean *rootfs*, final String *value*)

Parameter:

rootfs	as root file System.
value	as value of the file.

Dokumentation der Elementfunktionen

Boolean model.packages.Path.getRootfs ()

Rückgabe:

the rootfs

String model.packages.Path.getValue ()

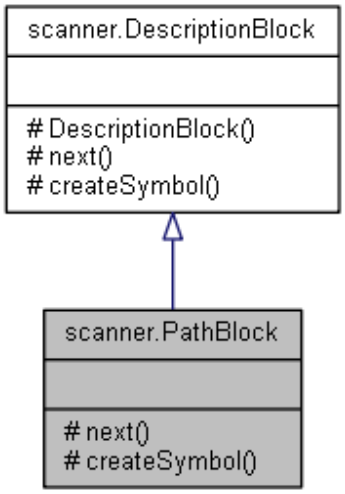
Rückgabe:
the value

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

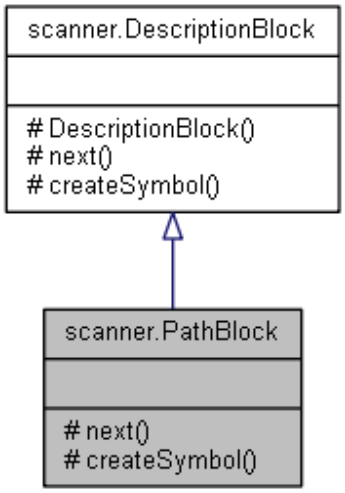
- workspace/ipfli4l13/src/model/packages/Path.java

scanner.PathBlock Klassenreferenz

scanner.PathBlockKlassendiagramm für scanner.PathBlock:



Zusammengehörigkeiten von scanner.PathBlock:



Geschützte Methoden

- scanner.DescriptionBlock next ()
- AbstractSymbol createSymbol (final String collectedContent, final Position position)

Ausführliche Beschreibung

This class represents the third block, the Path.

Autor:
Hendrik

Dokumentation der Elementfunktionen

AbstractSymbol scanner.PathBlock.createSymbol (final String *collectedContent*, final Position *position*)[protected], [virtual]

Create the new Symbol.

Parameter:

collectedContent	collected Content which is used for the Symbol.
position	the position of the collected content.

Rückgabe:
the new Symbol

Implementiert scanner.DescriptionBlock.

scanner.DescriptionBlock scanner.PathBlock.next ()[protected], [virtual]

set the next State.

Rückgabe:
the new State

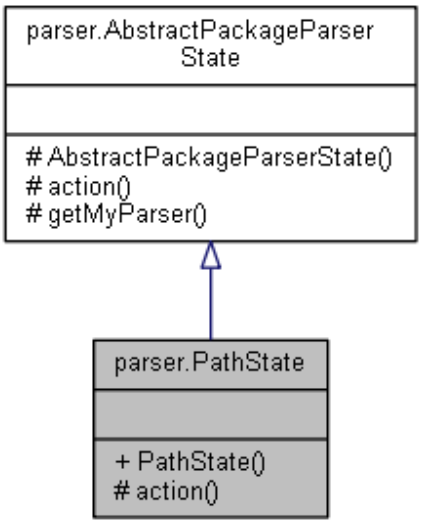
Implementiert scanner.DescriptionBlock.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

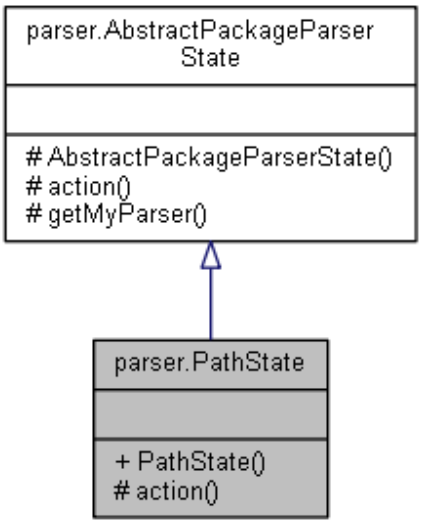
- workspace/ipfli4l13/src/scanner/PathBlock.java

parser.PathState Klassenreferenz

parser.PathStateKlassendiagramm für parser.PathState:



Zusammengehörigkeiten von parser.PathState:



Öffentliche Methoden

- PathState (final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule)

Geschützte Methoden

- AbstractPackageParserState **action** (final PackageDescriptionFileSymbol symbol, final Map< String, AbstractVariableDefinition > variableDef) throws InterruptedException, ParserException

Ausführliche Beschreibung

This class represents the Path State of the PackageDescriptionfileparser.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.PathState.PathState (final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*)

The constructor for the PathState.

Parameter:

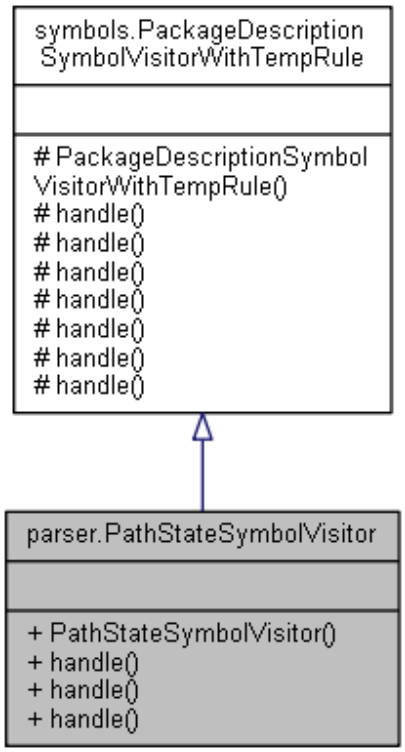
parser	for the DescriptionFileParser.
tempPackageRule	the temp PackageRule.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

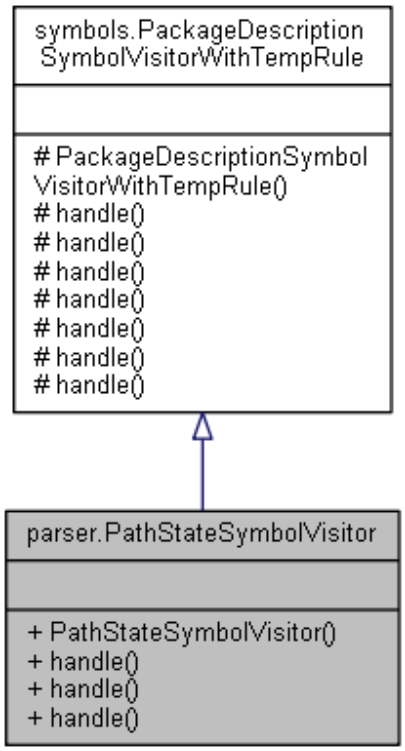
- workspace/ipfli4l13/src/parser/PathState.java

parser.PathStateSymbolVisitor Klassenreferenz

parser.PathStateSymbolVisitorKlassendiagramm für parser.PathStateSymbolVisitor:



Zusammengehörigkeiten von parser.PathStateSymbolVisitor:



Öffentliche Methoden

- PathStateSymbolVisitor ()
- AbstractPackageParserState **handle** (final CommentSymbol commentSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule)
- AbstractPackageParserState **handle** (final RowEndSymbol rowEndSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException
- AbstractPackageParserState **handle** (final OptionSymbol optionSymbol, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the Visitor for the PathStateSymbol.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.PathStateSymbolVisitor.PathStateSymbolVisitor ()

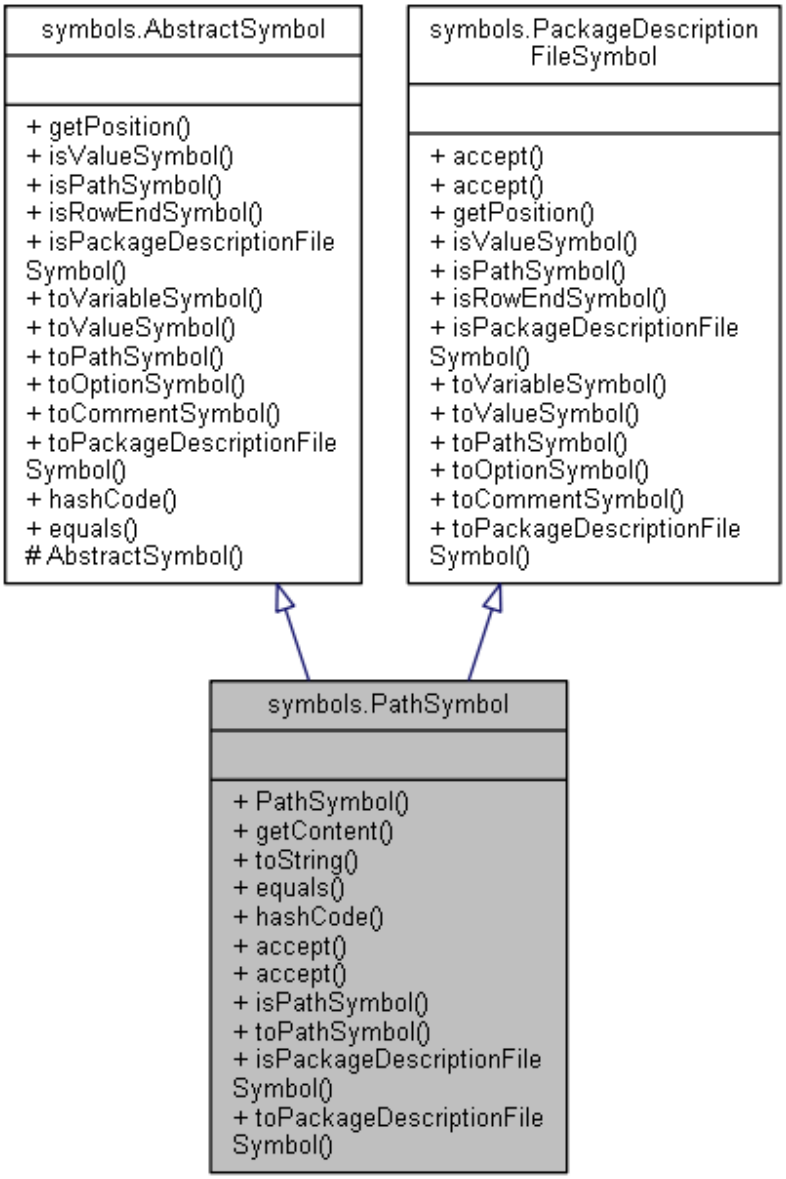
The constructor for the PathStateSymbolVisitor.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

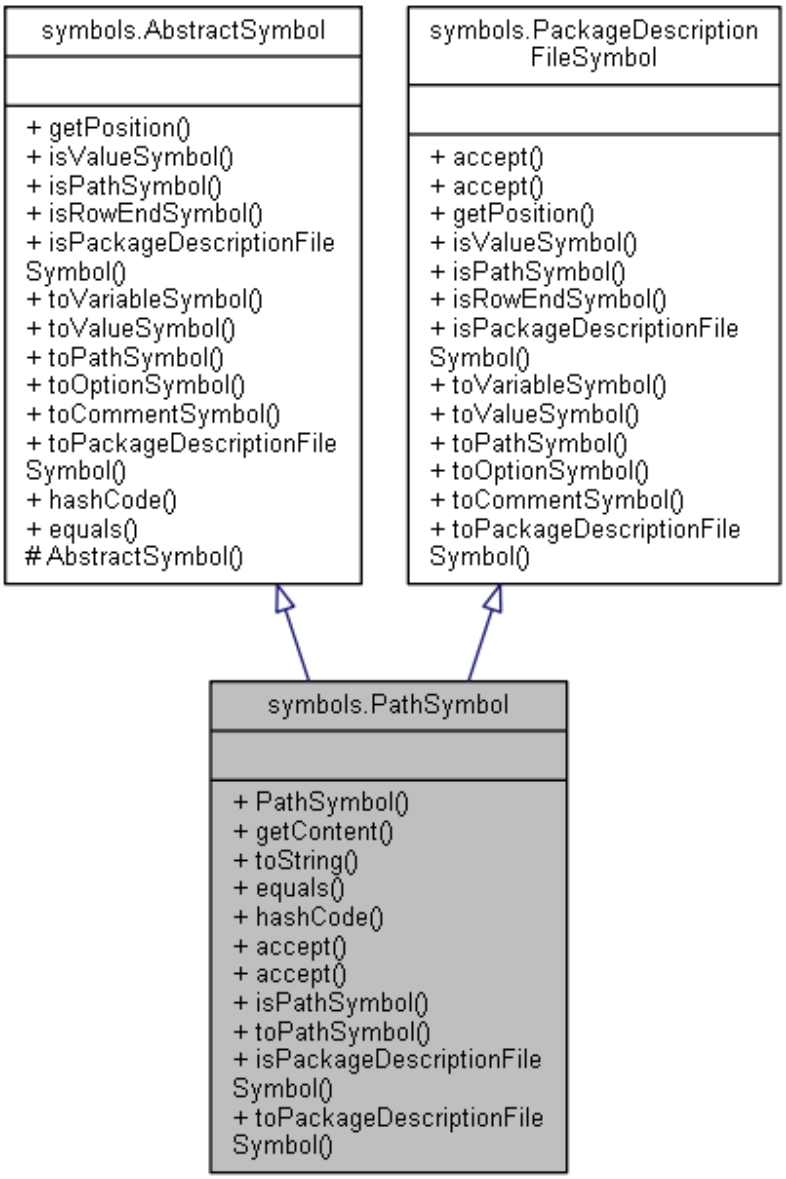
- workspace/ipfli4l13/src/parser/PathStateSymbolVisitor.java

symbols.PathSymbol Klassenreferenz

symbols.PathSymbolKlassendiagramm für symbols.PathSymbol:



Zusammengehörigkeiten von symbols.PathSymbol:



Öffentliche Methoden

- PathSymbol (final String content, final Position position)
- String getContent ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitor visitor, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitorWithTempRule visitor, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- Boolean isPathSymbol ()
- PathSymbol toPathSymbol ()

- Boolean isPackageDescriptionFileSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the path symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.PathSymbol.PathSymbol (final String *content*, final Position *position*)

This constructor instantiate the path symbol.

Parameter:

position	is the position of the data in the stream.
content	is the content of the PathState.

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.PathSymbol.accept (final PackageDescriptionSymbolVisitor *visitor*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

AbstractPackageParserState symbols.PathSymbol.accept (final PackageDescriptionSymbolVisitorWithTempRule *visitor*, final

PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.
tempPackageRule	is a temporary package rule.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

String symbols.PathSymbol.getContent ()

Rückgabe:

the content of the PathSymbol.

Boolean symbols.PathSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

Boolean symbols.PathSymbol.isPathSymbol ()

Rückgabe:

true, if the Symbol is a path symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

PackageDescriptionFileSymbol symbols.PathSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:

a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

PathSymbol symbols.PathSymbol.toPathSymbol ()

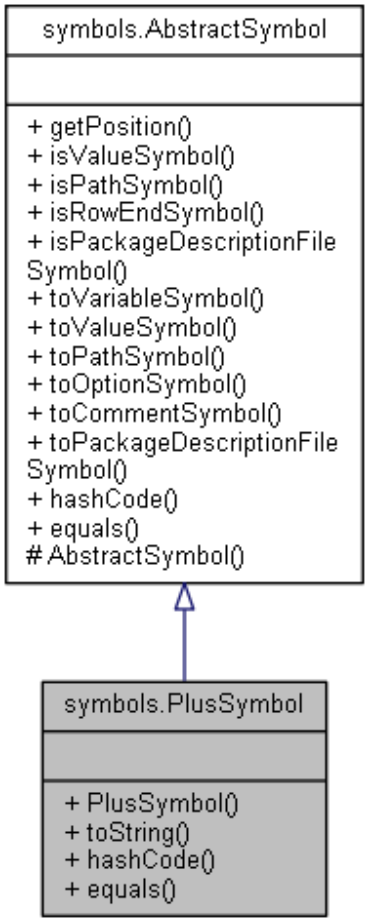
Rückgabe:
an error, if the symbol is not a path symbol
Implementiert symbols.PackageDescriptionFileSymbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

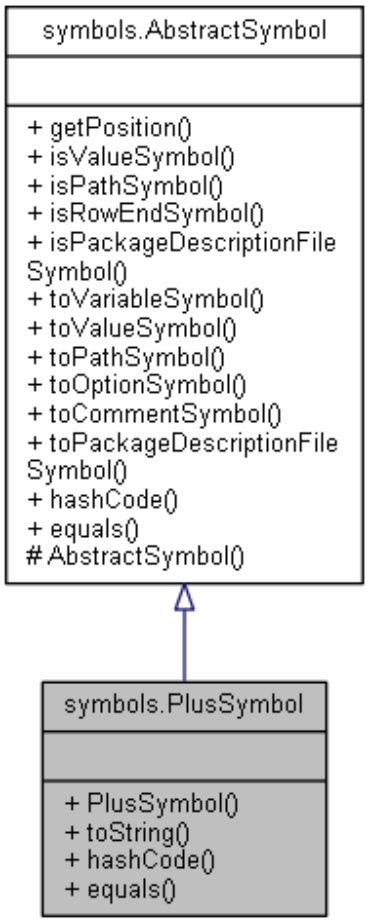
- workspace/ipfli4l13/src/symbols/PathSymbol.java

symbols.PlusSymbol Klassenreferenz

symbols.PlusSymbolKlassendiagramm für symbols.PlusSymbol:



Zusammengehörigkeiten von symbols.PlusSymbol:



Öffentliche Methoden

- PlusSymbol (final Position position)
- String **toString** ()
- int **hashCode** ()
- boolean **equals** (final Object obj)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the plussymbol (+).

Beschreibung der Konstruktoren und Destruktoren

symbols.PlusSymbol.PlusSymbol (final Position *position*)

The constructor instantiate the plussymbol.

Parameter:

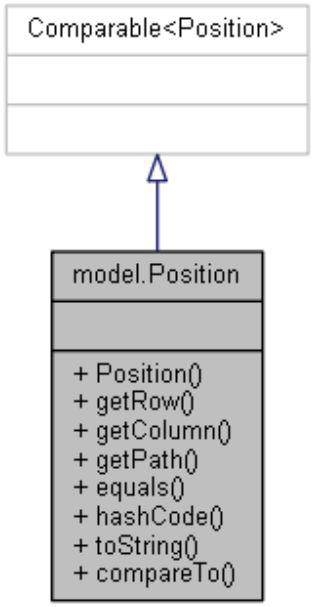
position	is the row and column of the data in the stream.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

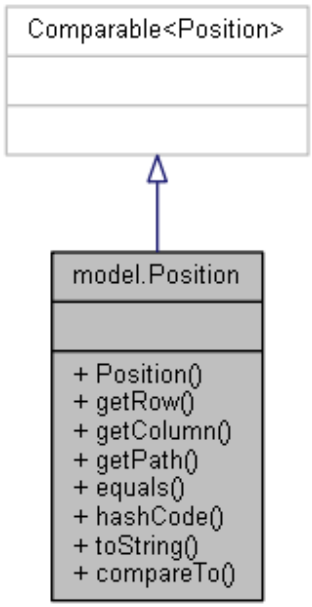
- workspace/ipfli4l13/src/symbols/PlusSymbol.java

model.Position Klassenreferenz

model.PositionKlassendiagramm für model.Position:



Zusammengehörigkeiten von model.Position:



Öffentliche Methoden

- **Position** (final Integer row, final Integer column, final String path)

- Integer getRow ()
- Integer getColumn ()
- String getPath ()
- boolean **equals** (final Object obj)
- int **hashCode** ()
- String **toString** ()
- int compareTo (final Position o)

Ausführliche Beschreibung

This class represents the position of the symbol.

Autor:

HFW410

Dokumentation der Elementfunktionen

int model.Position.compareTo (final Position *o*)

Compares the product(row, colomn) of two prodcts.

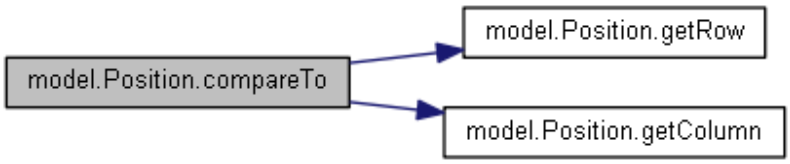
Parameter:

o	the position to compare.

Rückgabe:

returns int

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Integer model.Position.getColumn ()

Rückgabe:

column of the position.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

String model.Position.getPath ()

Rückgabe:
the path

Integer model.Position.getRow ()

Rückgabe:
row of the position.

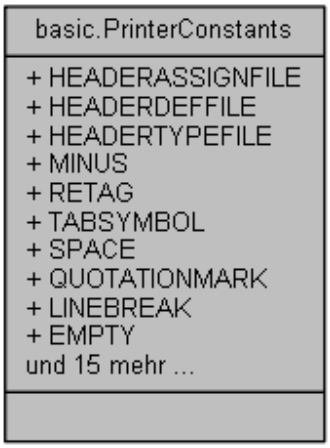
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/Position.java

basic.PrinterConstants Klassenreferenz

basic.PrinterConstantsZusammengehörigkeiten von basic.PrinterConstants:



Statische öffentliche Attribute

- static final String HEADERASSIGNFILE = ""
- static final String HEADERDEFFILE = ""
- static final String HEADERTYPEFILE = ""
- static final String MINUS = "-"
- static final String RETAG = "RE:"
- static final String TABSYMBOL = "\t"
- static final String SPACE = " "
- static final String QUOTATIONMARK = "\""
- static final String LINEBREAK = "\n"
- static final String EMPTY = ""
- static final String SINGLEQUOT = "'"
- static final String ENCODING = "UTF-8"
- static final String PATHFORPRINTING = "io/output/"
- static final String ASSIGNMENTPATH = "config/"
- static final String ASSIGNEND = ".txt"
- static final String DEFPATH = "check/"
- static final String DEFEND = ASSIGNEND
- static final String TYPEPATH = DEFPATH
- static final String TYPEEND = ".exp"
- static final String PRINTEXCEPTION = "The printer caused an Exception!"
- static final String BREAKOPEN = "("
- static final String BREAKCLOSE = ")"
- static final String TYPENOTEXISTEXC = "The Type does not exist: "
- static final String ANONVARERRORMESG = "No error Message available."
- static final String **PATHNOTAVAILABLAE**

Ausführliche Beschreibung

A Class for the String-Constants.

Autor:
HFW410RA - Philipp Rammos

Dokumentation der Datenelemente

final String basic.PrinterConstants.ANONVARERRORMESG = "No error Message available."[static]

the error message for an AnonVariableType with no error message.

final String basic.PrinterConstants.ASSIGNEND = ".txt"[static]

The fileending for printing for AssignmentFile.

final String basic.PrinterConstants.ASSIGNMENTPATH = "config/"[static]

The path for printing for AssignmentFile.

final String basic.PrinterConstants.BREAKCLOSE = ")"[static]

String representation for opening break).

final String basic.PrinterConstants.BREAKOPEN = "("[static]

String representation for opening break (.

final String basic.PrinterConstants.DEFEND = ASSIGNEND[static]

The fileending for printing for DefinitionFile.

final String basic.PrinterConstants.DEFPATH = "check/"[static]

The path for printing for DefinitiionFile.

final String basic.PrinterConstants.EMPTY = ""[static]

String representation for empty string.

final String basic.PrinterConstants.ENCODING = "UTF-8"[static]

The encoding for print-files.

final String basic.PrinterConstants.HEADERASSIGNFILE = ""[static]

String representation for the Header of AssignmentFile.

final String basic.PrinterConstants.HEADERDEFFILE = ""[static]

String representation for the Header of DefinitionFile.

final String basic.PrinterConstants.HEADERTYPEFILE = ""[static]

String representation for the Header of TypeFile.

final String basic.PrinterConstants.LINEBREAK = "\n"[static]

String representation for line break.

final String basic.PrinterConstants.MINUS = "-"[static]

String representation for "-".

final String basic.PrinterConstants.PATHFORPRINTING = "io/output/"[static]

The general path for printing.

final String basic.PrinterConstants.PATHNOTAVAILABLAE[static]

```
Initialisierung:=
            "The path of added Data was modified or removed, pls consider that the original
path exists: "
```

final String basic.PrinterConstants.PRINTEXCEPTION = "The printer caused an Exception!"[static]

The messge for exceptions caused by printer.

final String basic.PrinterConstants.QUOTATIONMARK = "\""[static]

String representation for the quotation mark "\" ".

final String basic.PrinterConstants.RETAG = "RE:"[static]

String representation for the re-tag "RE:".

final String basic.PrinterConstants.SINGLEQUOT = "'"[static]

String representation for single quotation symbol.

final String basic.PrinterConstants.SPACE = " "[static]

String representation for the space " ".

final String basic.PrinterConstants.TABSYMBOL = "\t"[static]

String representation for the tabsspace " ".

final String basic.PrinterConstants.TYPEEND = ".exp"[static]

The fileending for printing for TypeFile.

final String basic.PrinterConstants.TYPENOTEXISTEXC = "The Type does not exist: "[static]

the message for TYPEDOESNOTEXISTEXCEPTION.

final String basic.PrinterConstants.TYPEPATH = DEFPATH[static]

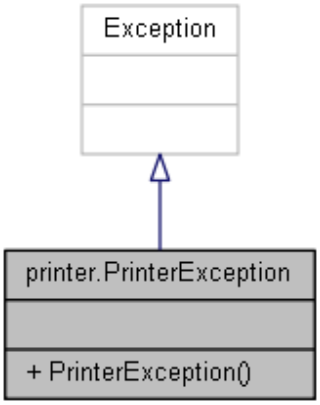
The path for printing for TypeFile.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

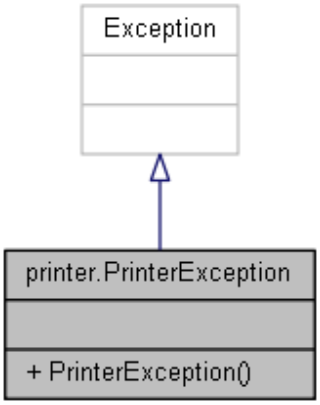
- workspace/ipfli4l13/src/basic/PrinterConstants.java

printer.PrinterException Klassenreferenz

printer.PrinterExceptionKlassendiagramm für printer.PrinterException:



Zusammengehörigkeiten von printer.PrinterException:



Öffentliche Methoden

- PrinterException (final String message, final Exception causeE)

Ausführliche Beschreibung

An Exception for all Exceptions caused by printing files.

Autor:

hfw410 - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

printer.PrinterException.PrinterException (final String *message*, final Exception *causeE*)

Constructor for creating an Exception for all Exceptions caused by printing files.

Parameter:

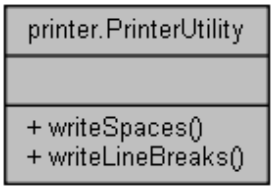
causeE	The original exception caused by java.
message	The message to set.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/printer/PrinterException.java

printer.PrinterUtility Klassenreferenz

printer.PrinterUtilityZusammengehörigkeiten von printer.PrinterUtility:



Öffentliche Methoden

- int writeSpaces (final StringBuffer stringBuffer, final int col, final Integer colPosition)
- int writeLineBreaks (final StringBuffer stringBuffer, final int row, final Integer rowPosition)

Ausführliche Beschreibung

UtilityClass for extracted Printer-Functions.

Autor:

Phil

Dokumentation der Elementfunktionen

int printer.PrinterUtility.writeLineBreaks (final StringBuffer *stringBuffer*, final int *row*, final Integer *rowPosition*)

Checks if a linebreak is needed and prints it into the writer.

Parameter:

row	the actual row in the file.
rowPosition	the row where we have to print.
stringBuffer	the given stringBuffer to print in.

Rückgabe:

the number of written lines.

int printer.PrinterUtility.writeSpaces (final StringBuffer *stringBuffer*, final int *col*, final Integer *colPosition*)

Checks if spaces are needed and prints it into the writer.

Parameter:

col	the actual col in the row.
colPosition	the col where we have to print.

stringBuffer	the given stringBuffer to print in.

Rückgabe:

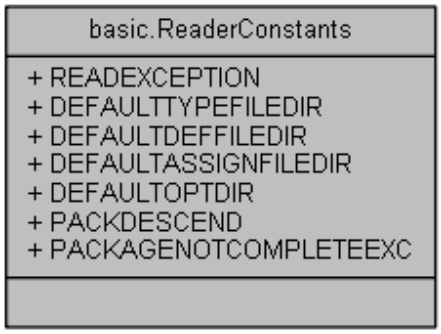
the number of written lines.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/printer/PrinterUtility.java

basic.ReaderConstants Klassenreferenz

basic.ReaderConstantsZusammengehörigkeiten von basic.ReaderConstants:



Statische öffentliche Attribute

- static final String READEXCEPTION = "An exception is caused by the reader!"
- static final String DEFAULTTYPEFILEDIR = "\\check\\"
- static final String DEFAULTDEFFILEDIR = DEFAULTTYPEFILEDIR
- static final String DEFAULTASSIGNFILEDIR = "\\config\\"
- static final String DEFAULTOPTDIR = "\\opt\\"
- static final String PACKDESCEND = ".txt"
- static final String **PACKAGENOTCOMPLETEEXC** = "The package is not complete."

Ausführliche Beschreibung

Autor:
Phil

Dokumentation der Datenelemente

final String basic.ReaderConstants.DEFAULTASSIGNFILEDIR = "\\config\\"[static]

String representation of default assignmentfile-directory.

final String basic.ReaderConstants.DEFAULTDEFFILEDIR = DEFAULTTYPEFILEDIR[static]

String representation of default deffile-directory.

final String basic.ReaderConstants.DEFAULTOPTDIR = "\\opt\\"[static]

String representation of default opt-directory.

final String basic.ReaderConstants.DEFAULTTYPEFILEDIR = "\\check\\"[static]

String representation of default typefile-directory.

final String basic.ReaderConstants.PACKDESCEND = ".txt"[static]

The file-ending for reading for package description files.

final String basic.ReaderConstants.READEXCEPTION = "An exception is caused by the reader!"[static]

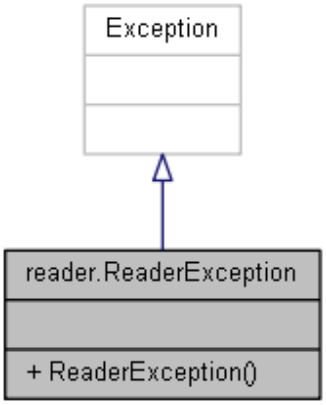
String representation for the reader Exception.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

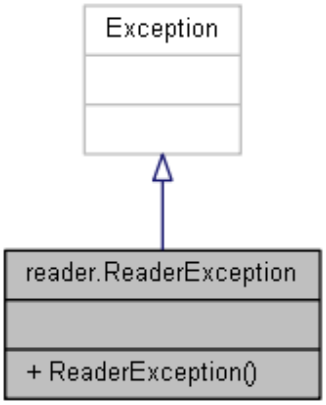
- workspace/ipfli4l13/src/basic/ReaderConstants.java

reader.ReaderException Klassenreferenz

reader.ReaderExceptionKlassendiagramm für reader.ReaderException:



Zusammengehörigkeiten von reader.ReaderException:



Öffentliche Methoden

- ReaderException (final String message, final Exception causeE)

Ausführliche Beschreibung

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

reader.ReaderException.ReaderException (final String *message*, final Exception *causeE*)

Constructor for creating an Exception for all Exceptions caused by reading files.

Parameter:

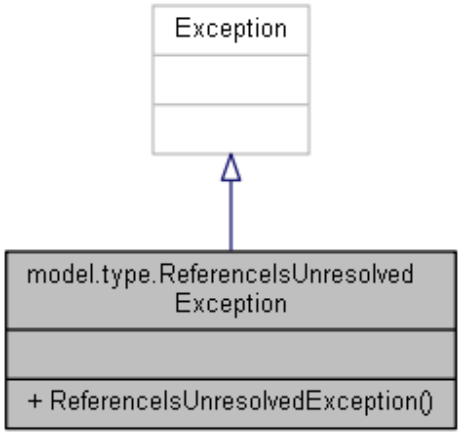
causeE	The original exception caused by java.
message	The message to set.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

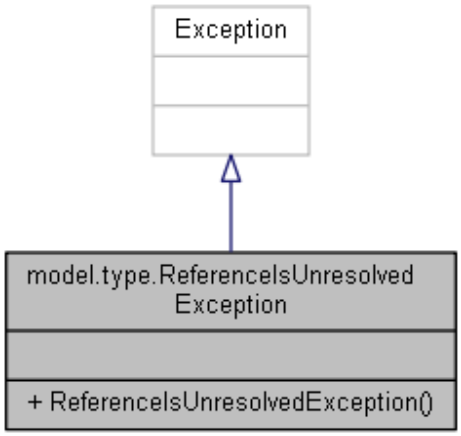
- workspace/ipfli4l13/src/reader/ReaderException.java

model.type.ReferenceIsUnresolvedException Klassenreferenz

model.type.ReferenceIsUnresolvedExceptionKlassendiagramm für model.type.ReferenceIsUnresolvedException:



Zusammengehörigkeiten von model.type.ReferenceIsUnresolvedException:



Öffentliche Methoden

- ReferenceIsUnresolvedException (final String errorMessage)

Ausführliche Beschreibung

Autor:
HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.ReferenceIsUnresolvedException.ReferenceIsUnresolvedException (final String *errorMessage*)

Constructor, just calls super(errorMesage).

Parameter:

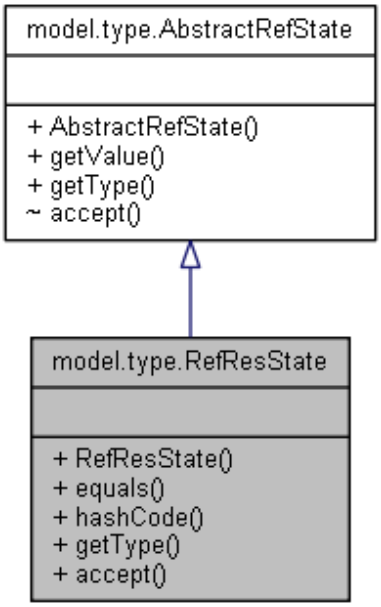
errorMessage	The message for the exception.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

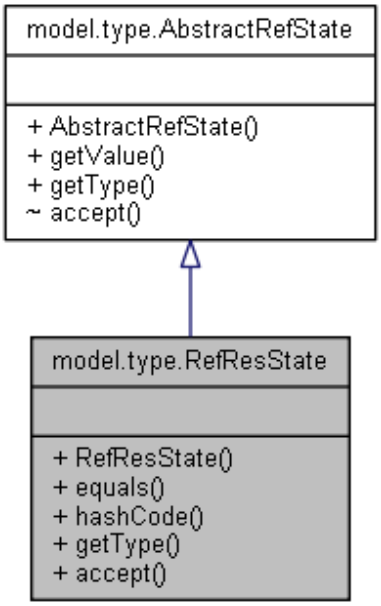
- workspace/ipfli4l13/src/model/type/ReferenceIsUnresolvedException.java

model.type.RefResState Klassenreferenz

model.type.RefResStateKlassendiagramm für model.type.RefResState:



Zusammengehörigkeiten von model.type.RefResState:



Öffentliche Methoden

- RefResState (final NamedVariableType type)

- boolean **equals** (final Object obj)
- int **hashCode** ()
- NamedVariableType getType ()
- void accept (final RefStateVisitor vis)

Ausführliche Beschreibung

A state that represents that a reference is resolved.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.RefResState.RefResState (final NamedVariableType *type*)

Constructor, tries to set the field. Checks if it would be a cycle.

Parameter:

type	The type to set.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Dokumentation der Elementfunktionen

void model.type.RefResState.accept (final RefStateVisitor *vis*)`[virtual]`

accept method for visitor pattern.

Parameter:

vis	the visitor.

Ausnahmebehandlung:

TypeDoesNotExistException	if a part is not resolvable

Implementiert model.type.AbstractRefState.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

NamedVariableType model.type.RefResState.getType ()[virtual]

Returns the field type. No side effects.

Rückgabe:

this.type

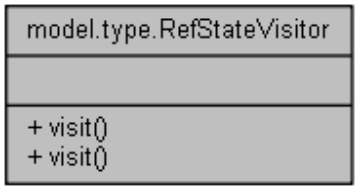
Implementiert model.type.AbstractRefState.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/type/RefResState.java

model.type.RefStateVisitor Schnittstellenreferenz

model.type.RefStateVisitorZusammengehörigkeiten von model.type.RefStateVisitor:



Öffentliche Methoden

- void visit (RefResState state)
- void visit (RefUnResState state) throws TypeDoesNotExistException

---

Ausführliche Beschreibung

RefStateVisitor, Visitor-Pattern.

Autor:

Phil

Dokumentation der Elementfunktionen

void model.type.RefStateVisitor.visit (RefResState *state*)

visit method for Visitor-pattern.

Parameter:

state	the state

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

void model.type.RefStateVisitor.visit (RefUnResState *state*) throws TypeDoesNotExistException

visit method for Visitor-pattern.

Parameter:

state	the state

Ausnahmebehandlung:

TypeDoesNotE-	if a part is not resolvable.

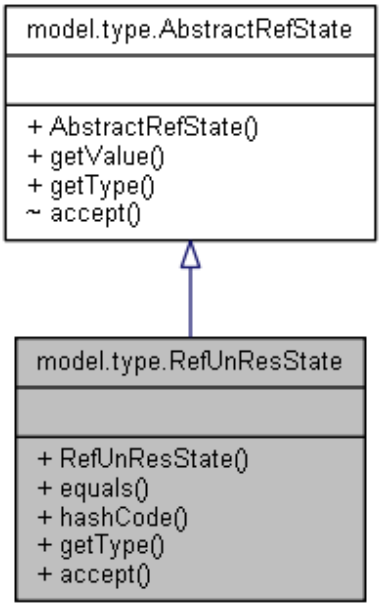
xistException	

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

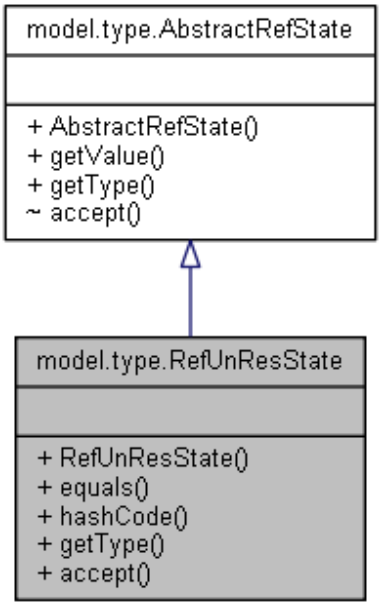
- workspace/ipfli4l13/src/model/type/RefStateVisitor.java

model.type.RefUnResState Klassenreferenz

model.type.RefUnResStateKlassendiagramm für model.type.RefUnResState:



Zusammengehörigkeiten von model.type.RefUnResState:



Öffentliche Methoden

- RefUnResState (final String value)

- boolean **equals** (final Object obj)
- int **hashCode** ()
- NamedVariableType getType () throws ReferenceIsUnresolvedException
- void accept (final RefStateVisitor vis) throws TypeDoesNotExistException

---

Ausführliche Beschreibung

A state that represents that a reference is unresolved.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.RefUnResState.RefUnResState (final String *value*)

Represents the stringrepresentation of the reference. Constructor, just sets fields. No side effects.

Parameter:

value	The value to set.

Dokumentation der Elementfunktionen

void model.type.RefUnResState.accept (final RefStateVisitor *vis*) throws TypeDoesNotExistException `[virtual]`

accept method for visitor pattern.

Parameter:

vis	the visitor.

Ausnahmebehandlung:

TypeDoesNotExistException	if a part is not resolvable

Implementiert model.type.AbstractRefState.

NamedVariableType model.type.RefUnResState.getType () throws ReferenceIsUnresolvedException `[virtual]`

Returns the field type, if it is an RefResState, else an exception is thrown.

Rückgabe:

this.type

Ausnahmebehandlung:

ReferenceIsUnresolvedException	If the state isnt a RefResState.

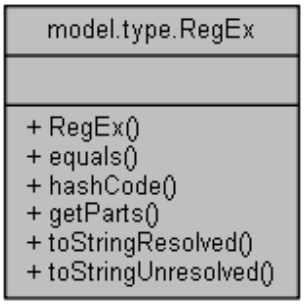
Implementiert model.type.AbstractRefState.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/type/RefUnResState.java

model.type.RegEx Klassenreferenz

model.type.RegExZusammengehörigkeiten von model.type.RegEx:



Öffentliche Methoden

- RegEx (final java.util.List< RegExPart > parts)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- java.util.List< RegExPart > getParts ()
- String toStringResolved () throws ReferenceIsUnresolvedException
- String toStringUnresolved ()

Ausführliche Beschreibung

Representation of a regular expression.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.RegEx.RegEx (final java.util.List< RegExPart > *parts*)

Constructor, just sets the fields. No side effects.

Parameter:

parts	The parts to set.

Dokumentation der Elementfunktionen

java.util.List<RegExPart> model.type.RegEx.getParts ()

Returns the field parts. no side effects.

Rückgabe:

this.part

String model.type.RegEx.toStringResolved () throws ReferenceIsUnresolvedException

Returns the resolved string of the regexpart , withOUT references like (RE:LABEL).

Rückgabe:

string

Ausnahmebehandlung:

ReferenceIsUnresolvedException	If a part is unresolved.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

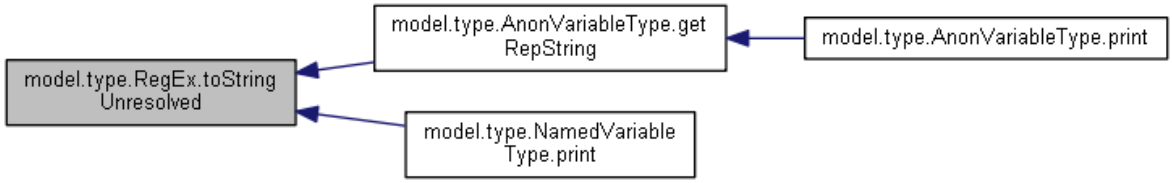
String model.type.RegEx.toStringUnresolved ()

Returns the unresolved string of the regexpart, with references like (RE:LABEL).

Rückgabe:

string

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

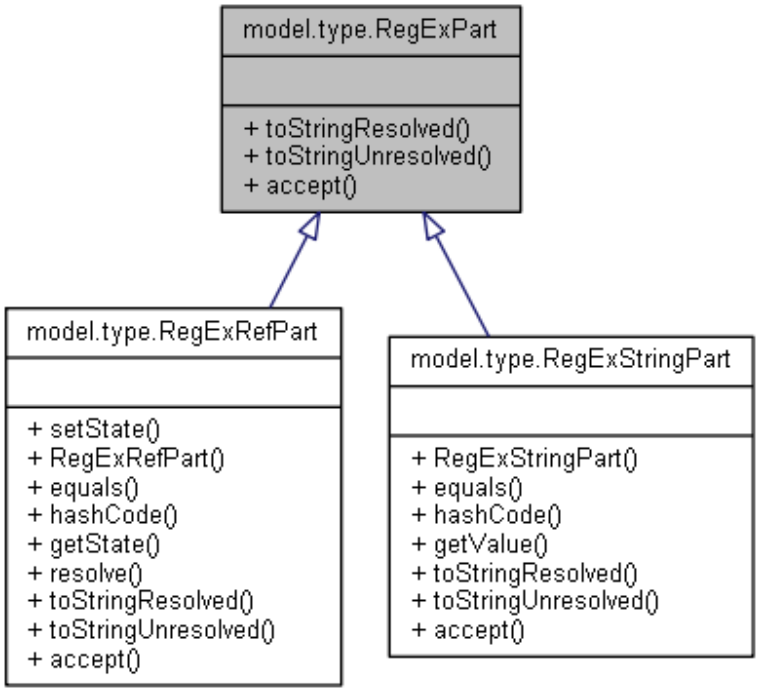


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

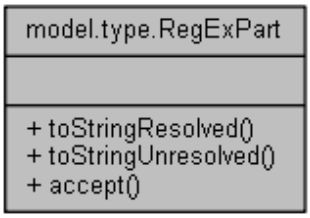
- workspace/ipfli4l13/src/model/type/RegEx.java

model.type.RegExPart Schnittstellenreferenz

model.type.RegExPartKlassendiagramm für model.type.RegExPart:



Zusammengehörigkeiten von model.type.RegExPart:



Öffentliche Methoden

- String toStringResolved () throws ReferenceIsUnresolvedException
- String toStringUnresolved ()
- void accept (RegExPartVisitor vis) throws TypeDoesNotExistException

---

Ausführliche Beschreibung

Autor:

HFW410RA - Philipp Rammos

Dokumentation der Elementfunktionen

void model.type.RegExPart.accept (RegExPartVisitor *vis*) throws TypeDoesNotExistException

accept method for visitor pattern.

Parameter:

vis	the visitor.

Ausnahmebehandlung:

TypeDoesNotExistException	if a part is not resolvable

String model.type.RegExPart.toStringResolved () throws ReferenceIsUnresolvedException

Returns the resolved string of the regexpart , withOUT references like (RE:LABEL).

Rückgabe:

string

Ausnahmebehandlung:

ReferenceIsUnresolvedException	If a part is unresolved.

Implementiert in model.type.RegExRefPart und model.type.RegExStringPart.

String model.type.RegExPart.toStringUnresolved ()

Returns the unresolved string of the regexpart, with references like (RE:LABEL).

Rückgabe:

string

Implementiert in model.type.RegExRefPart und model.type.RegExStringPart.

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/model/type/RegExPart.java

model.type.RegExPartVisitor Schnittstellenreferenz

model.type.RegExPartVisitorZusammengehörigkeiten von model.type.RegExPartVisitor:

Öffentliche Methoden

- void visit (final RegExStringPart regexstringpart)
- void visit (final RegExRefPart regexrefpart) throws TypeDoesNotExistException

---

Ausführliche Beschreibung

Visitor for RegExPart, Visitor-Pattern.

Autor:

HFW410RA - Philipp Rammos

Dokumentation der Elementfunktionen

void model.type.RegExPartVisitor.visit (final RegExStringPart *regexstring-part*)

visit method for visitor pattern.

Parameter:

regexstringpart	Part to visit.

void model.type.RegExPartVisitor.visit (final RegExRefPart *regexrefpart*) throws TypeDoesNotExistException

visit method for visitor pattern.

Parameter:

regexrefpart	part to visit.

Ausnahmebehandlung:

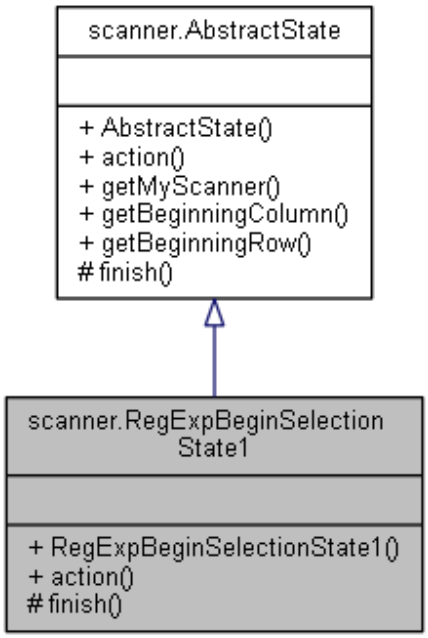
TypeDoesNotExistException	if a part is not resolvable

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

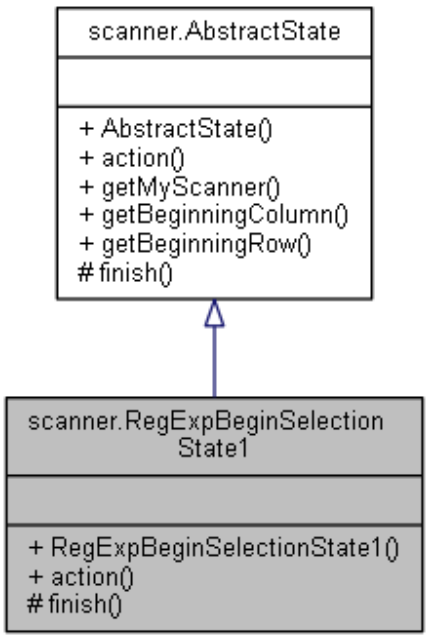
- workspace/ipfli4l13/src/model/type/RegExPartVisitor.java

scanner.RegExpBeginSelectionState1 Klassenreferenz

scanner.RegExpBeginSelectionState1Klassendiagramm für scanner.RegExpBeginSelectionState1:



Zusammengehörigkeiten von scanner.RegExpBeginSelectionState1:



Öffentliche Methoden

- RegExpBeginSelectionState1 (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)
- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath)

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

This class represents the RegExpBeginSelectionState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.RegExpBeginSelectionState1.RegExpBeginSelectionState1 (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

This constructor initialize the RegExpBeginSelectionState.

Parameter:

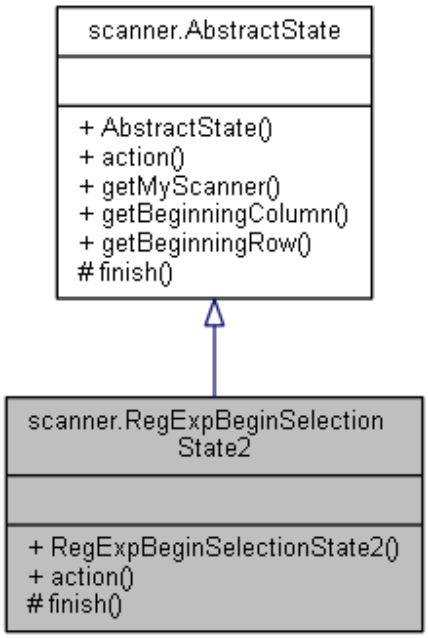
myScanner	is the used Scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the beginning row of the symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

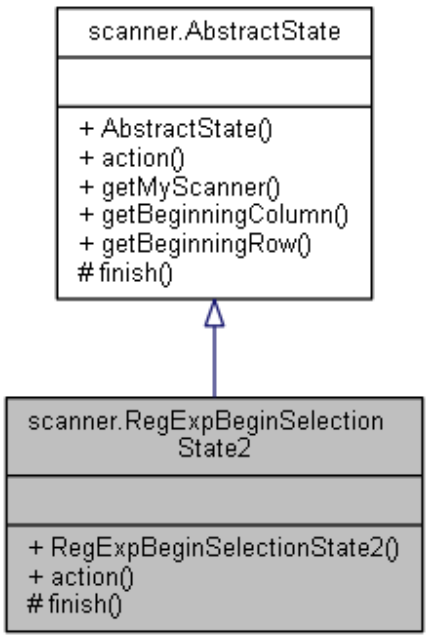
- workspace/ipfli4l13/src/scanner/RegExpBeginSelectionState1.java

scanner.RegExpBeginSelectionState2 Klassenreferenz

scanner.RegExpBeginSelectionState2Klassendiagramm für scanner.RegExpBeginSelectionState2:



Zusammengehörigkeiten von scanner.RegExpBeginSelectionState2:



Öffentliche Methoden

- RegExpBeginSelectionState2 (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)
- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

This class represents the 2. BeginSelectionState for the Regular Expressions.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.RegExpBeginSelectionState2.RegExpBeginSelectionState2 (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

initialize the RegExpBeginSelectionState2.

Parameter:

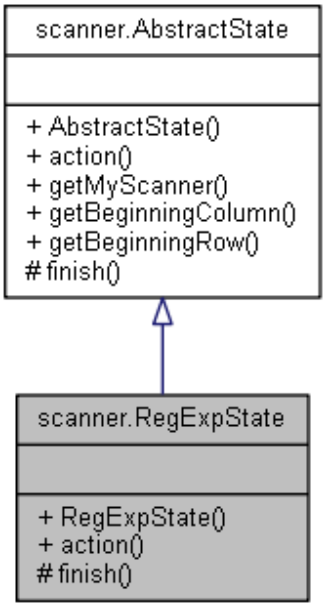
myScanner	is the used Scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the beginning row of the symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

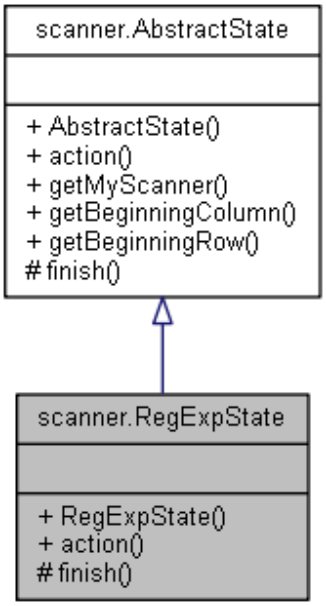
- workspace/ipfli4l13/src/scanner/RegExpBeginSelectionState2.java

scanner.RegExpState Klassenreferenz

scanner.RegExpStateKlassendiagramm für scanner.RegExpState:



Zusammengehörigkeiten von scanner.RegExpState:



Öffentliche Methoden

- RegExpState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)

- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

This class represents the regular Expression.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.RegExpState.RegExpState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

The constructor to initialize the RegExState.

Parameter:

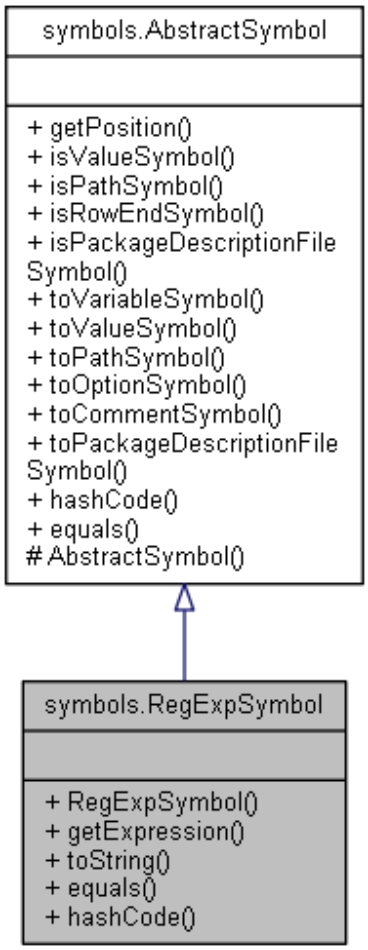
myScanner	the used Scanner
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the beginning row of the symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

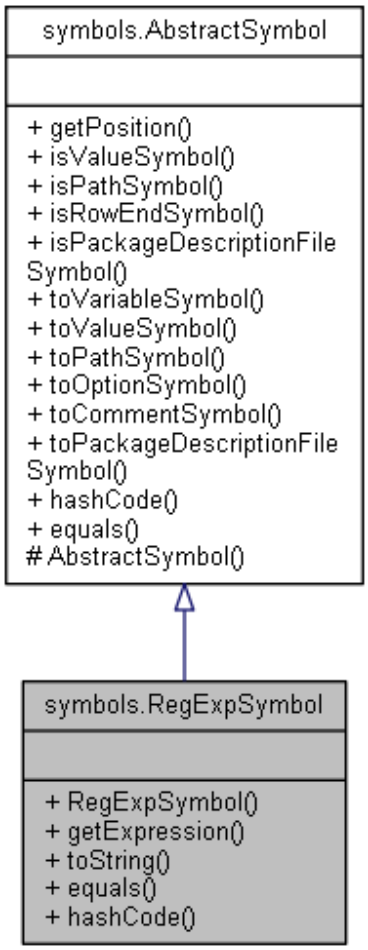
- workspace/ipfli4l13/src/scanner/RegExpState.java

symbols.RegExpSymbol Klassenreferenz

symbols.RegExpSymbolKlassendiagramm für symbols.RegExpSymbol:



Zusammengehörigkeiten von symbols.RegExpSymbol:



Öffentliche Methoden

- RegExpSymbol (final String ausdruck, final Position position)
- String getExpression ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the Regular Expression Symbol.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

symbols.RegExpSymbol.RegExpSymbol (final String *ausdruck*, final Position *position*)

This constructor initialize the RegExpSymbol.

Parameter:

ausdruck	of the Symbol.
position	of the expression in thedatastream

Dokumentation der Elementfunktionen

String symbols.RegExpSymbol.getExpression ()

Rückgabe:

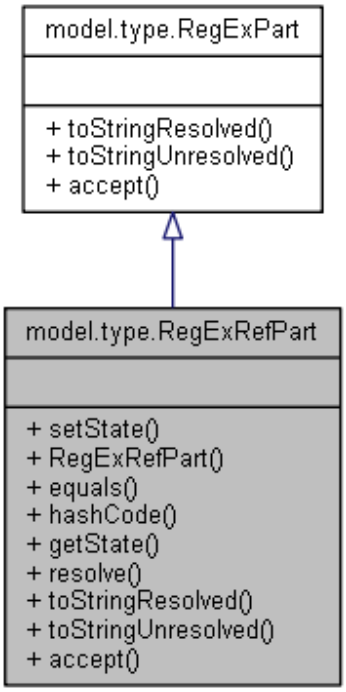
the expression

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

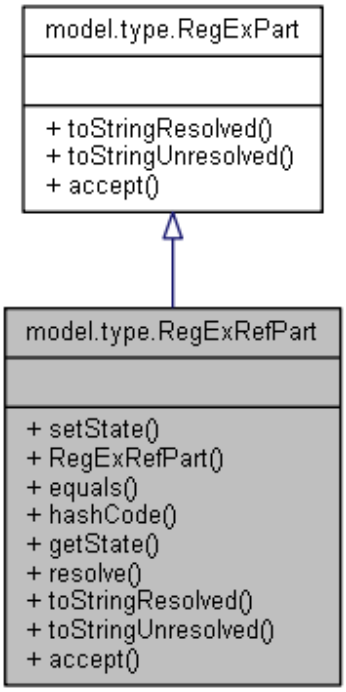
- workspace/ipfli4l13/src/symbols/RegExpSymbol.java

model.type.RegExRefPart Klassenreferenz

model.type.RegExRefPartKlassendiagramm für model.type.RegExRefPart:



Zusammengehörigkeiten von model.type.RegExRefPart:



Öffentliche Methoden

- void setState (final AbstractRefState state)
- RegExRefPart (final AbstractRefState state)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- AbstractRefState getState ()
- void resolve (final NamedVariableType variableType) throws ReferenceIsUnresolvedException
- String toStringResolved () throws ReferenceIsUnresolvedException
- String toStringUnresolved ()
- void **accept** (final RegExPartVisitor vis) throws TypeDoesNotExistException

---

Ausführliche Beschreibung

Represents a "re:typename" as a part of a regular expression.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.RegExRefPart.RegExRefPart (final AbstractRefState *state*)

Constructor, just sets fields. No side effects.

Parameter:

state	The state to set.

Dokumentation der Elementfunktionen

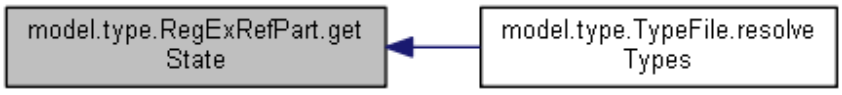
AbstractRefState model.type.RegExRefPart.getState ()

Returns the field state. No side effects.

Rückgabe:

this.state

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



void model.type.RegExRefPart.resolve (final NamedVariableType *variableType*) throws ReferenceIsUnresolvedException

Changes the state to RefResState with the given type. Dont use it, its called by TypeFile. Use typeFile.resolveTypes() instead.

Parameter:

variableType	The Type to set.

Ausnahmebehandlung:

ReferenceIsUnresolvedException	If a part of the type is unresolved.

void model.type.RegExRefPart.setState (final AbstractRefState *state*)

Sets a new state.

Parameter:

state	The state to set.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

String model.type.RegExRefPart.toStringResolved () throws ReferenceIsUnresolvedException

Returns the resolved string of the regexpart , withOUT references like (RE:LABEL).

Rückgabe:

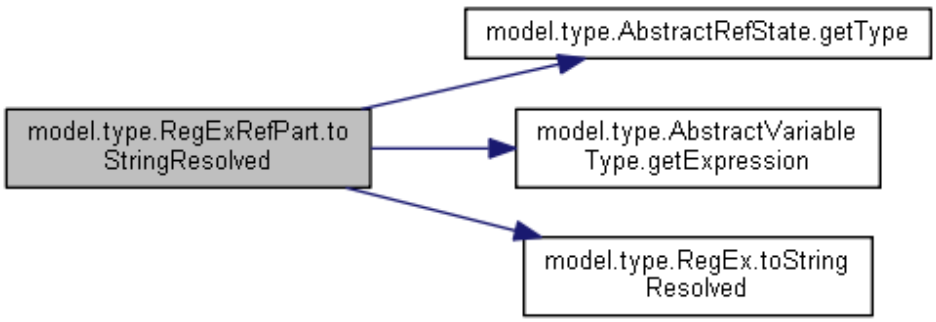
string

Ausnahmebehandlung:

ReferenceIsUnresolvedException	If a part is unresolved.

Implementiert model.type.RegExPart.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



String model.type.RegExRefPart.toStringUnresolved ()

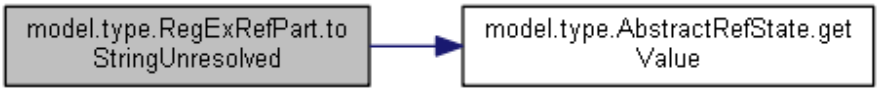
Returns the unresolved string of the regexpart, with references like (RE:LABEL).

Rückgabe:

string

Implementiert model.type.RegExPart.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

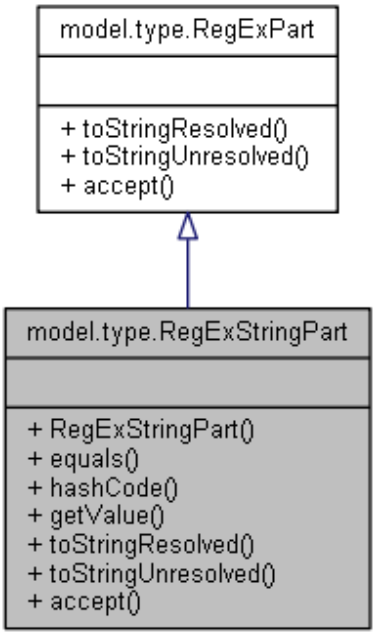


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

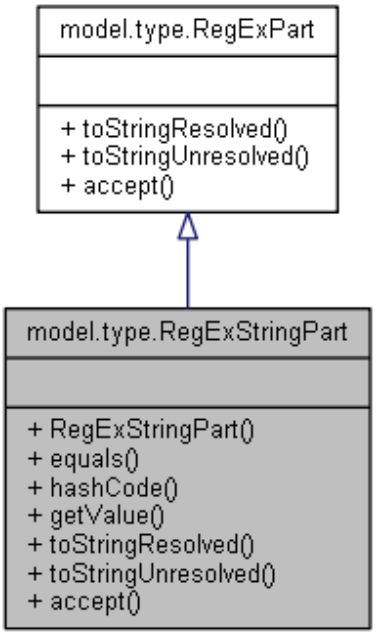
- workspace/ipfli4l13/src/model/type/RegExRefPart.java

model.type.RegExStringPart Klassenreferenz

model.type.RegExStringPartKlassendiagramm für model.type.RegExStringPart:



Zusammengehörigkeiten von model.type.RegExStringPart:



Öffentliche Methoden

- RegExStringPart (final String value)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- String getValue ()
- String toStringResolved ()
- String toStringUnresolved ()
- void **accept** (final RegExPartVisitor vis)

Ausführliche Beschreibung

Represents a string part of a regular expression.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.RegExStringPart.RegExStringPart (final String *value*)

Constructor, just sets the field. No side effects.

Parameter:

value	string part to set.

Dokumentation der Elementfunktionen

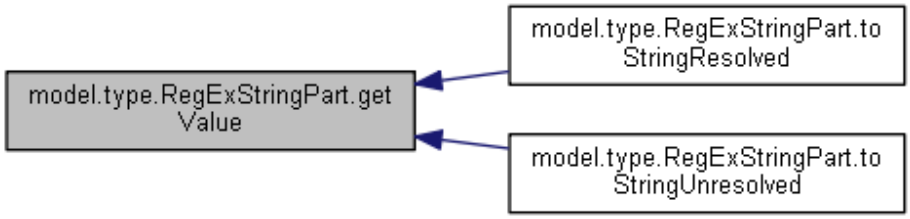
String model.type.RegExStringPart.getValue ()

Returns the field value. No side effects.

Rückgabe:

this.value

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



String model.type.RegExStringPart.toStringResolved ()

Returns the resolved string of the regexpart , withOUT references like (RE:LABEL).

Rückgabe:

string

Ausnahmebehandlung:

ReferenceIsUnresolvedException	If a part is unresolved.

Implementiert model.type.RegExPart.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

String model.type.RegExStringPart.toStringUnresolved ()

Returns the unresolved string of the regexpart, with references like (RE:LABEL).

Rückgabe:

string

Implementiert model.type.RegExPart.

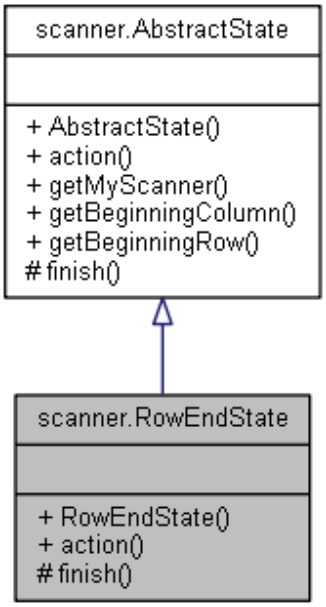
Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

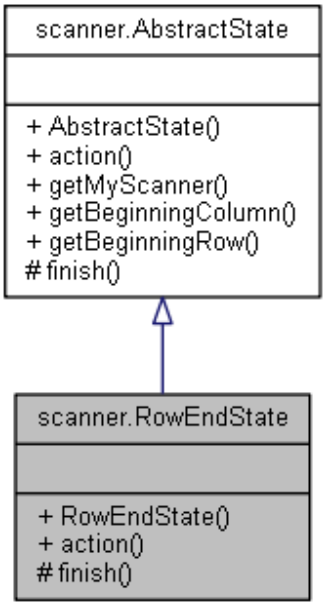
- workspace/ipfli4l13/src/model/type/RegExStringPart.java

scanner.RowEndState Klassenreferenz

scanner.RowEndStateKlassendiagramm für scanner.RowEndState:



Zusammengehörigkeiten von scanner.RowEndState:



Öffentliche Methoden

- RowEndState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)

- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath)

Ausführliche Beschreibung

state, which handle the new line sign.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

scanner.RowEndState.RowEndState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

creates a RowEndState.

Parameter:

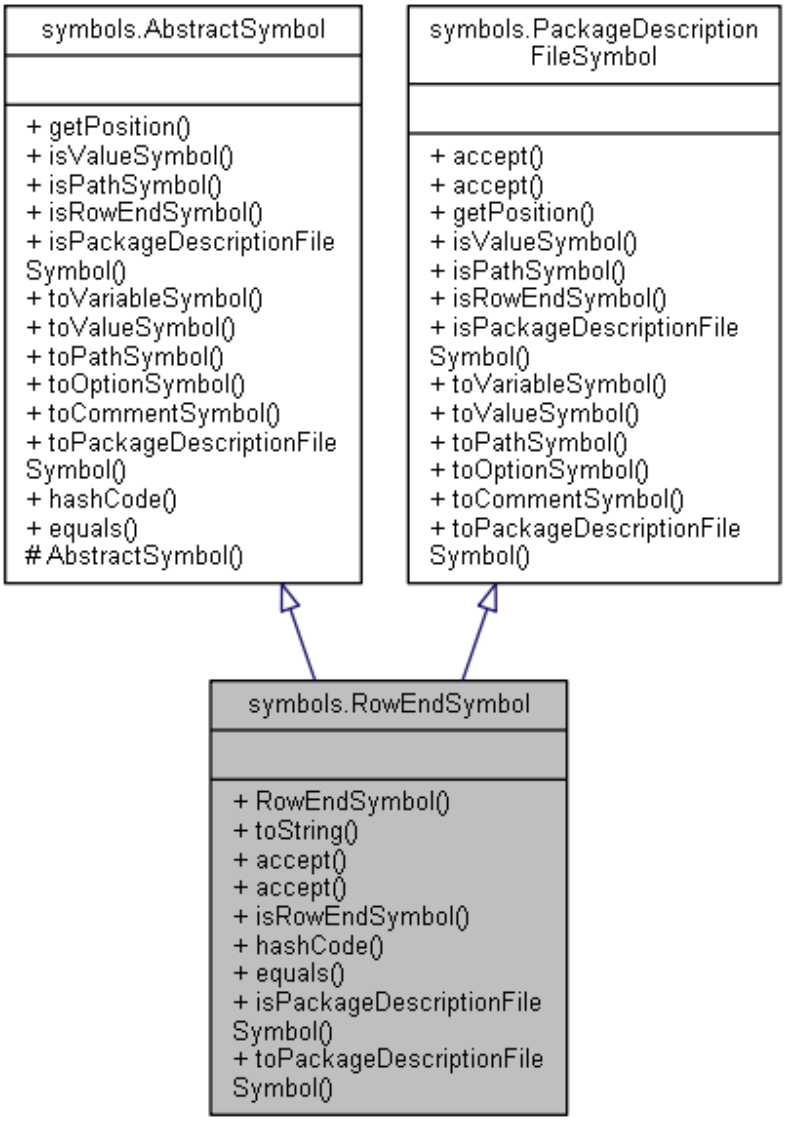
myScanner	is the scanner, which owns the state.
beginningColumn	is the first column of the symbol.
beginningRow	is the first row of the symbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

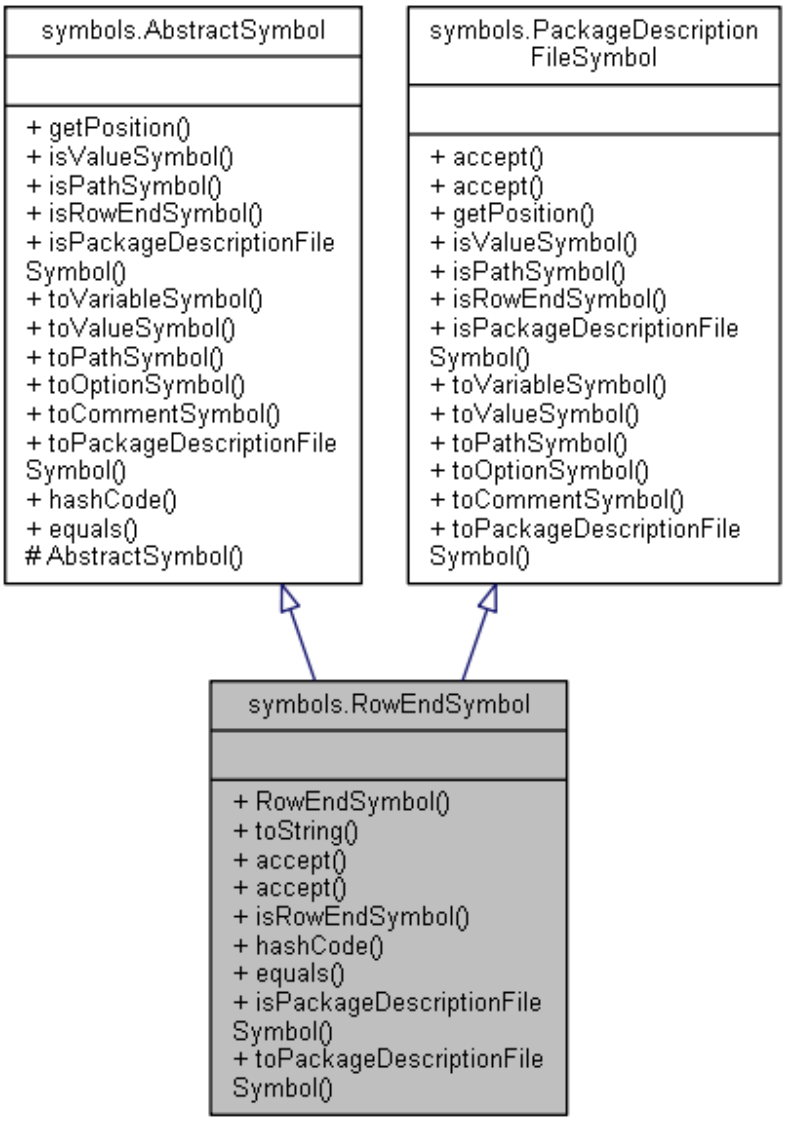
- workspace/ipfli4l13/src/scanner/RowEndState.java

symbols.RowEndSymbol Klassenreferenz

symbols.RowEndSymbolKlassendiagramm für symbols.RowEndSymbol:



Zusammengehörigkeiten von symbols.RowEndSymbol:



Öffentliche Methoden

- RowEndSymbol (final Position position)
- String **toString** ()
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitor visitor, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitorWithTempRule visitor, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- Boolean isRowEndSymbol ()
- int **hashCode** ()
- boolean **equals** (final Object obj)
- Boolean isPackageDescriptionFileSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the row end symbol. This is the end of a row.

Beschreibung der Konstruktoren und Destruktoren

symbols.RowEndSymbol.RowEndSymbol (final Position *position*)

This is the constructor for the row end symbol.

Parameter:

position	is the position of the data in the stream.

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.RowEndSymbol.accept (final PackageDescriptionSymbolVisitor *visitor*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

AbstractPackageParserState symbols.RowEndSymbol.accept (final PackageDescriptionSymbolVisitorWithTempRule *visitor*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackage-Rule*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.
tempPackageRule	is a temporary package rule.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

Boolean symbols.RowEndSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

Boolean symbols.RowEndSymbol.isRowEndSymbol ()

Rückgabe:

true, if the Symbol is a row end symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

PackageDescriptionFileSymbol symbols.RowEndSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:

a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.

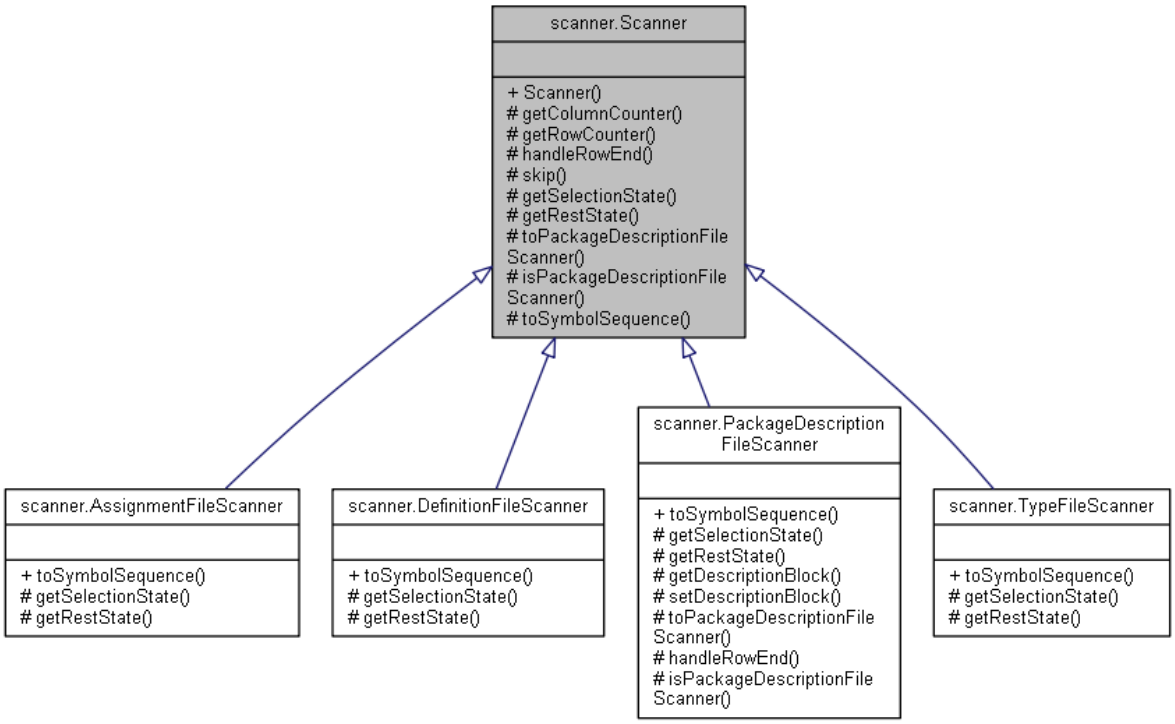
Implementiert symbols.PackageDescriptionFileSymbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

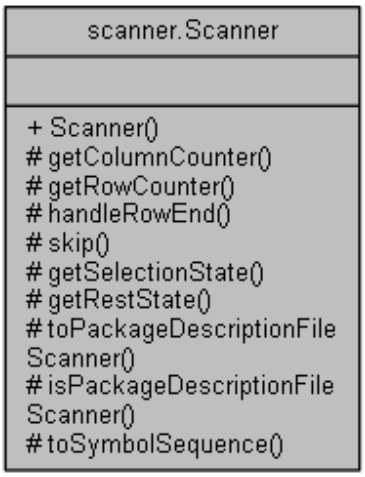
- workspace/ipfli4l13/src/symbols/RowEndSymbol.java

scanner.Scanner Klassenreferenz

scanner.ScannerKlassendiagramm für scanner.Scanner:



Zusammengehörigkeiten von scanner.Scanner:



Öffentliche Methoden

- Scanner (final Buffer< AbstractSymbol > buffer)

Geschützte Methoden

- Integer getColumnCounter ()
- Integer getRowCounter ()
- void handleRowEnd ()
- void skip ()
- abstract AbstractState getSelectionState ()
- abstract AbstractState getRestState (Integer beginningRow, Integer beginningColumn)
- PackageDescriptionFileScanner toPackageDescriptionFileScanner ()
- Boolean isPackageDescriptionFileScanner ()

Geschützte, statische Methoden

- static void toSymbolSequence (final InputStream input, final String dataPath, final Scanner scanner) throws ScannerException, InterruptedException, IOException

---

Ausführliche Beschreibung

This interface represents the Scanner.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.Scanner.Scanner (final Buffer< AbstractSymbol > *buffer*)

This is the constructoer to set the attributes.

Parameter:

buffer	is the symbolbuffer.

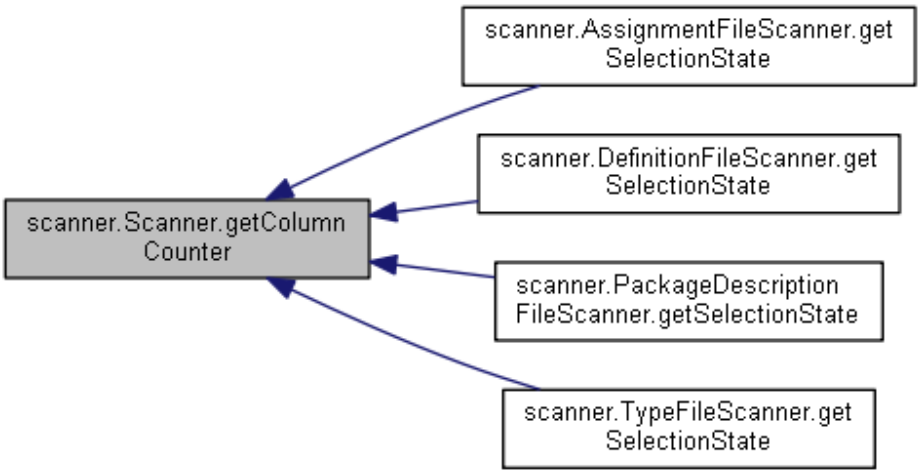
Dokumentation der Elementfunktionen

Integer scanner.Scanner.getColumnCounter ()[protected]

Rückgabe:

the CulumnCounter

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



abstract AbstractState scanner.Scanner.getRestState (Integer *beginningRow*, Integer *beginningColumn*)`[protected]`, `[pure virtual]`

Parameter:

beginningRow	is the beginning row of the symbol
beginningColumn	is the beginning column of the symbol

Rückgabe:

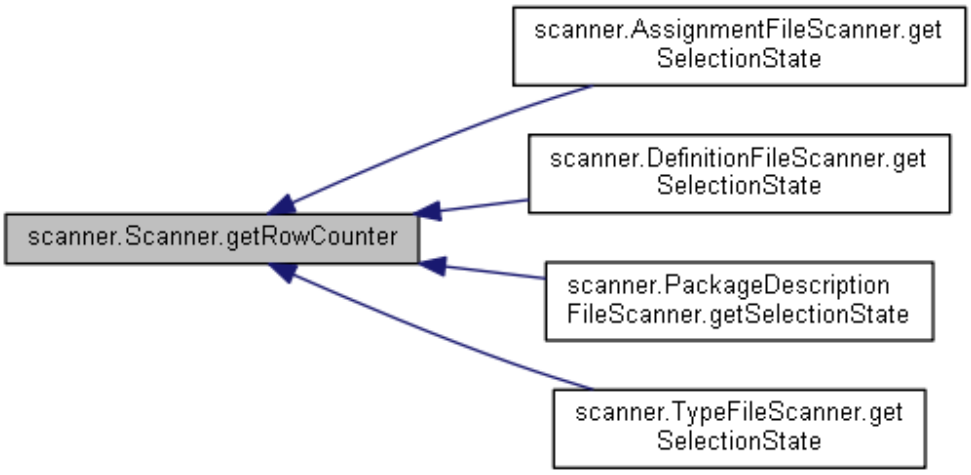
AbstractState a reststate of the special scanner with the given beginning row and column.

Integer scanner.Scanner.getRowCounter ()`[protected]`

Rückgabe:

the RowCounter

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



abstract AbstractState scanner.Scanner.getSelectionState ()[protected], [pure virtual]

Rückgabe:

the SelectionState of the special scanner.

Implementiert in scanner.PackageDescriptionFileScanner, scanner.AssignmentFileScanner, scanner.DefinitionFileScanner und scanner.TypeFileScanner.

void scanner.Scanner.handleRowEnd ()[protected]

this method increase the RowCounter.

Boolean scanner.Scanner.isPackageDescriptionFileScanner ()[protected]

Rückgabe:

true if the Scanner is a PackageDescriptionFilescanner.

void scanner.Scanner.skip ()[protected]

This method skips the first character and increments the Column Counter.

PackageDescriptionFileScanner scanner.Scanner.toPackageDescriptionFileScanner ()[protected]

Rückgabe:

an error, if the symbol is not a descriptionfile scanner

static void scanner.Scanner.toSymbolSequence (final InputStream *input*, final String *dataPath*, final Scanner *scanner*) throws ScannerException, InterruptedException, IOException `[static]`, `[protected]`

this methode will be used to call the Scanner.

Parameter:

input	is the input stream.
dataPath	is the path of the file, which will be scanned.
scanner	is the Scanner, which is used.

Ausnahmebehandlung:

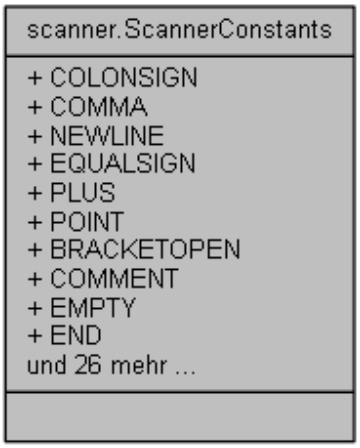
ScannerException	is the Exception from all scanners.
InterruptedException	if there is any problem with the buffer.
IOException	from the Stream.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/scanner/Scanner.java

scanner.ScannerConstants Klassenreferenz

scanner.ScannerConstantsZusammengehörigkeiten von scanner.ScannerConstants:



Statische öffentliche Attribute

- static final Character COLONSIGN = ':'
- static final Character COMMA = ','
- static final Character NEWLINE = '\n'
- static final Character EQUALSIGN = '='
- static final Character PLUS = '+'
- static final Character POINT = '.'
- static final Character BRACKETOPEN = '['
- static final String COMMENT = "Comment: "
- static final String EMPTY = "Empty Symbol"
- static final String END = "End"
- static final String ROWEND = "Row end"
- static final String STRING = "String: "
- static final String IDENTIFIER = "Identifier: "
- static final String UNKNOWN = "Unknown: "
- static final String REGEXP = "RE: "
- static final String EMPTYSTRING = ""
- static final String COMMASPACE = ", "
- static final String BUFFER = "Buffer: "
- static final String VARIABLE = "Variable: "
- static final String VALUE = "Value: "
- static final String PATH = "Path: "
- static final String OPTION = "Option: "
- static final String NO_STREAM_EXCEPTION = "The given Inputstream is null!"
- static final Character COMMENTSYMBOL = '#'
- static final Character SPACE = ' '
- static final Integer FIRST_POSITION = 0
- static final String POSITION = "Position: "
- static final Character PERCENTAGE = '%'
- static final Character UNDERSCORE = '_'
- static final String WTE_VARIABLE = "Unexpected symbol! Expected a variable symbol!"
- static final String WTE_VALUE = "Unexpected symbol! Expected a value symbol!"

- static final String WTE_PATH = "Unexpected symbol! Expected a path symbol!"
- static final String WTE_OPTION = "Unexpected symbol! Expected an option symbol!"
- static final String WTE_COMMENT = "Unexpected symbol! Expected a comment symbol!"
- static final String WTE_PACKAGESYMBOL
- static final String WTE_DESCRIPTIONFILESCANNER

Ausführliche Beschreibung

This class represents the constants of the scanner.

Autor:

HFW410

Dokumentation der Datenelemente

final Character scanner.ScannerConstants.BRACKETOPEN = '['[static]

This is the BracketOpen([) as Character.

final String scanner.ScannerConstants.BUFFER = "Buffer: "[static]

This is the Beginning of a Buffer.

final Character scanner.ScannerConstants.COLONSIGN = ':'[static]

This is a colonsign as Character (:).

final Character scanner.ScannerConstants.COMMA = ','[static]

This is a comma as Character (,).

final String scanner.ScannerConstants.COMMASPACE = ", "[static]

This is the Text of a Comma.

final String scanner.ScannerConstants.COMMENT = "Comment: "[static]

This is the Beginning of a comment as String.

final Character scanner.ScannerConstants.COMMENTSYMBOL = '#'[static]

This is the commentsymbol as Character.

final String scanner.ScannerConstants.EMPTY = "Empty Symbol"[static]

This is the text of an empty symbol as String.

final String scanner.ScannerConstants.EMPTYSTRING = ""[static]

This is the text of a EmptyString.

final String scanner.ScannerConstants.END = "End"[static]

This is the text of the end symbol as String.

final Character scanner.ScannerConstants.EQUALSIGN = '='[static]

This is a equalsign as Character (=).

final Integer scanner.ScannerConstants.FIRST_POSITION = 0[static]

The integer of the first position in the datastream.

final String scanner.ScannerConstants.IDENTIFIER = "Identifier: "[static]

This is the Beginning of an Identifier Symbol.

final Character scanner.ScannerConstants.NEWLINE = '\n'[static]

This is a new line as Character (
).

final String scanner.ScannerConstants.NO_STREAM_EXCEPTION = "The given Inputstream is null!"[static]

This is a text of a scannerexcption. It shows, the inputstram is null.

final String scanner.ScannerConstants.OPTION = "Option: "[static]

This is the beginning of a option symbol.

final String scanner.ScannerConstants.PATH = "Path: "[static]

This is the beginning of a path symbol.

final Character scanner.ScannerConstants.PERCENTAGE = '%'[static]

This is the Character for the percentage symbol.

final Character scanner.ScannerConstants.PLUS = '+'[static]

This is the PLUS as Character.

final Character scanner.ScannerConstants.POINT = '.'[static]

This is the POINT as Character.

final String scanner.ScannerConstants.POSITION = "Position: "[static]

String for the Position.

final String scanner.ScannerConstants.REGEXP = "RE: "[static]

This is the Beginning of a Reference.

final String scanner.ScannerConstants.ROWEND = "Row end"[static]

This is the text of the row end symbol as String.

final Character scanner.ScannerConstants.SPACE = ' '[static]

This is the Space as Character.

final String scanner.ScannerConstants.STRING = "String: "[static]

This is the Beginning of an Identifier Symbol.

final Character scanner.ScannerConstants.UNDERSCORE = '_'[static]

This is the Character for the underscore symbol.

final String scanner.ScannerConstants.UNKNOWN = "Unknown: "[static]

This is the Beginning of an unknown Symbol.

final String scanner.ScannerConstants.VALUE = "Value: "[static]

This is the beginning of a value symbol.

final String scanner.ScannerConstants.VARIABLE = "Variable: "[static]

This is the beginning of a variable symbol.

final String scanner.ScannerConstants.WTE_COMMENT = "Unexpected symbol! Expected a comment symbol!"[static]

This is the error message for the non-comment symbol.

final String scanner.ScannerConstants.WTE_DESCRIPTIONFILESCANNER[static]

```
Initialisierung:=
            "Unexpected SCANNER! Expected a description file scanner!"
```

This is the error message for the non-DescriptionFileScanner.

final String scanner.ScannerConstants.WTE_OPTION = "Unexpected symbol! Expected an option symbol!"[static]

This is the error message for a non-option symbol.

final String scanner.ScannerConstants.WTE_PACKAGESYMBOL[static]

```
Initialisierung:=
```

```
"Unexpected symbol! Expected a package description file symbol!"
```

This is the error message for the non-package description file symbol.

final String scanner.ScannerConstants.WTE_PATH = "Unexpected symbol! Expected a path symbol!"[static]

This is the error message for a non-path symbol.

final String scanner.ScannerConstants.WTE_VALUE = "Unexpected symbol! Expected a value symbol!"[static]

This is the error message for a non-value symbol.

final String scanner.ScannerConstants.WTE_VARIABLE = "Unexpected symbol! Expected a variable symbol!"[static]

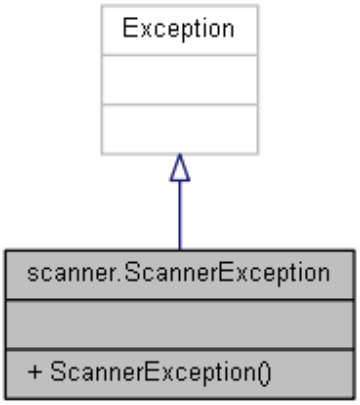
This is the error message for a non-variable symbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

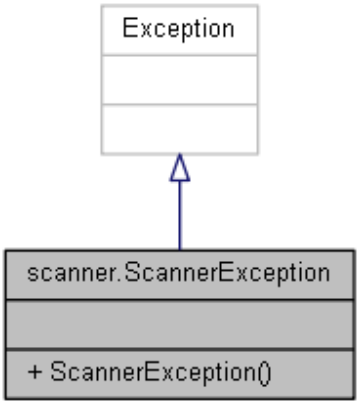
- workspace/ipfli4l13/src/scanner/ScannerConstants.java

scanner.ScannerException Klassenreferenz

scanner.ScannerExceptionKlassendiagramm für scanner.ScannerException:



Zusammengehörigkeiten von scanner.ScannerException:



Öffentliche Methoden

- ScannerException (final String text)

Ausführliche Beschreibung

This class represents all Exception from scanner.

Autor:

Hendrik

Beschreibung der Konstruktoren und Destruktoren

scanner.ScannerException.ScannerException (final String *text*)

This constructor initialize the ScannerException.

Parameter:

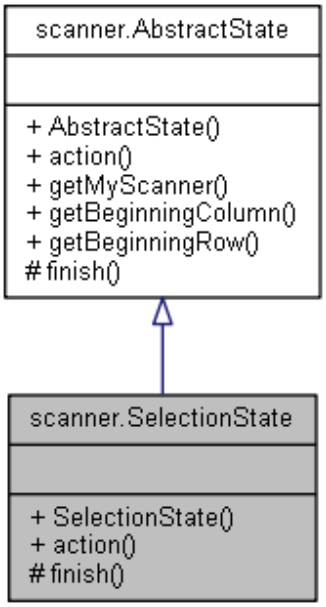
text	is the text of the exception.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

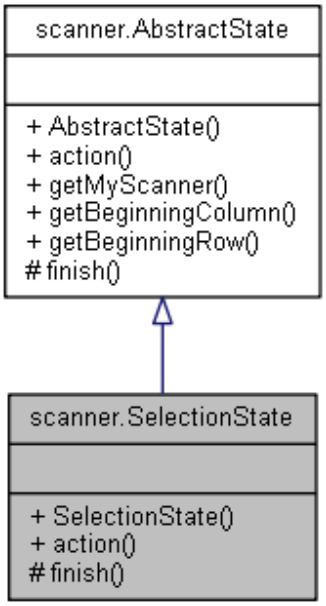
- workspace/ipfli4l13/src/scanner/ScannerException.java

scanner.SelectionState Klassenreferenz

scanner.SelectionStateKlassendiagramm für scanner.SelectionState:



Zusammengehörigkeiten von scanner.SelectionState:



Öffentliche Methoden

- SelectionState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)

- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath)

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

This class represents the SelectionState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.SelectionState.SelectionState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

This constructor initialize the new TypeFileSelectionState.

Parameter:

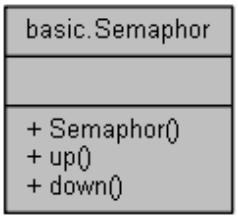
myScanner	is the used scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the beginning row of the symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/scanner/SelectionState.java

basic.Semaphor Klassenreferenz

basic.SemaphorZusammengehörigkeiten von basic.Semaphor:



Öffentliche Methoden

- Semaphor (final Integer count)
- synchronized void up ()
- synchronized void down () throws InterruptedException

Ausführliche Beschreibung

The Semaphore to lock or unlock a attribute.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

basic.Semaphor.Semaphor (final Integer *count*)

The constructor for the semaphore.

Parameter:

count	the counter for the semaphore.

Dokumentation der Elementfunktionen

synchronized void basic.Semaphor.down () throws InterruptedException

lock the attribute.

Ausnahmebehandlung:

InterruptedException	if it failed.

synchronized void basic.Semaphor.up ()

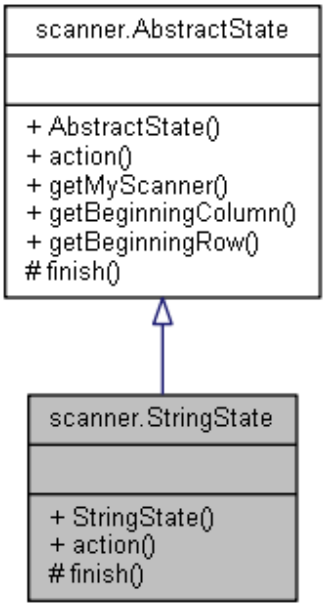
unlock the attribute.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

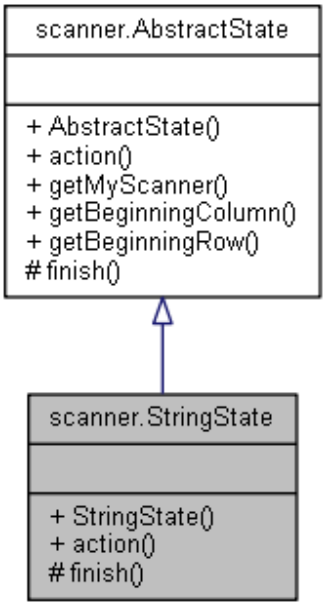
- workspace/ipfli4l13/src/basic/Semaphor.java

scanner.StringState Klassenreferenz

scanner.StringStateKlassendiagramm für scanner.StringState:



Zusammengehörigkeiten von scanner.StringState:



Öffentliche Methoden

- StringState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow, final Character quote)
- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

---

Ausführliche Beschreibung

This class represents the StringState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.StringState.StringState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*, final Character *quote*)

initialize the StringState.

Parameter:

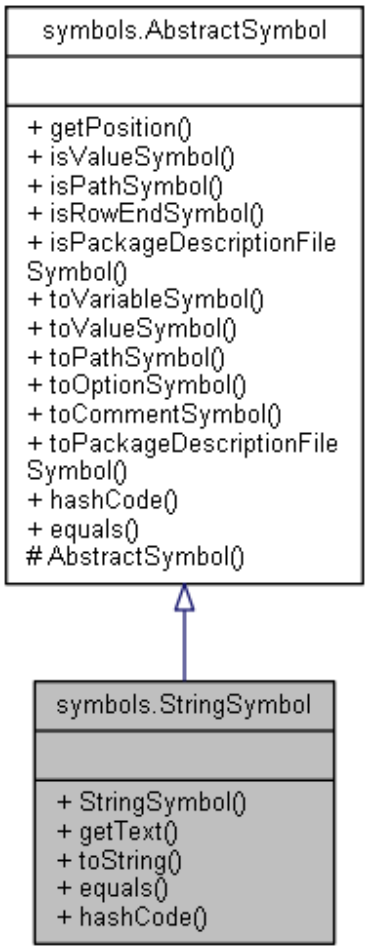
myScanner	is the used scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the beginning row of the symbol
quote	is the sign, which initated the string

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

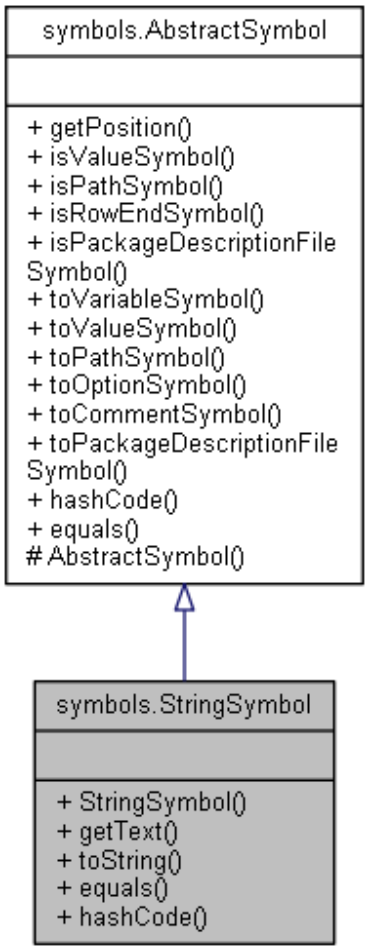
- workspace/ipfli4l13/src/scanner/StringState.java

symbols.StringSymbol Klassenreferenz

symbols.StringSymbolKlassendiagramm für symbols.StringSymbol:



Zusammengehörigkeiten von symbols.StringSymbol:



Öffentliche Methoden

- StringSymbol (final String text, final Position position)
- String getText ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents String symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.StringSymbol.StringSymbol (final String *text*, final Position *position*)

This constructor initialize the String Symbol.

Parameter:

text	is the text of the String.
position	is the row and column of the data in the stream.

Dokumentation der Elementfunktionen

String symbols.StringSymbol.getText ()

This method returns the attribute text.

Rückgabe:

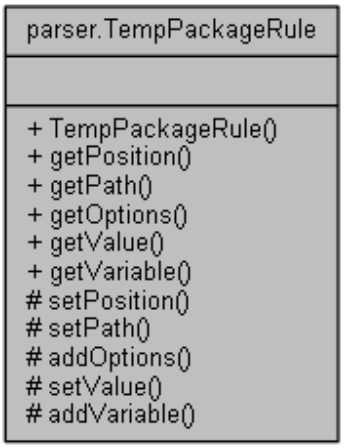
the text.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/symbols/StringSymbol.java

parser.TempPackageRule Klassenreferenz

parser.TempPackageRuleZusammengehörigkeiten von parser.TempPackageRule:



Öffentliche Methoden

- TempPackageRule ()
- Position getPosition ()
- Path getPath ()
- List< Option > getOptions ()
- RegEx getValue ()
- List< AbstractVariableDefinition > getVariable ()

Geschützte Methoden

- void setPosition (final Position position)
- void setPath (final Path path)
- void addOptions (final Option option)
- void setValue (final RegEx value)
- void addVariable (final AbstractVariableDefinition variable)

Ausführliche Beschreibung

is the temporary object for the PackageRule.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

parser.TempPackageRule.TempPackageRule ()

creates a new TempPackageRule.

Dokumentation der Elementfunktionen

void parser.TempPackageRule.addOptions (final Option *option*)[protected]

adds an Option.

Parameter:

option	is the Option, which should be added.

void parser.TempPackageRule.addVariable (final AbstractVariableDefinition *variable*)[protected]

sets the VariableDefinition.

Parameter:

variable	is the VariableDefinition.

List<Option> parser.TempPackageRule.getOptions ()

Rückgabe:

the List of Options.

Path parser.TempPackageRule.getPath ()

Rückgabe:

the Path of the PackageRule.

Position parser.TempPackageRule.getPosition ()

Rückgabe:

the position of the PackageRule.

RegEx parser.TempPackageRule.getValue ()

Rückgabe:

the value of the PackageRule.

List<AbstractVariableDefinition> parser.TempPackageRule.getVariable ()

Rückgabe:

the VariableDefinition of the PackageRule.

void parser.TempPackageRule.setPath (final Path *path*)[protected]

set the Path of the PackageRule.

Parameter:

path	the Path.

void parser.TempPackageRule.setPosition (final Position *position*)[protected]

sets the Position.

Parameter:

position	is the Position.

void parser.TempPackageRule.setValue (final RegEx *value*)[protected]

set the value of the package description file.

Parameter:

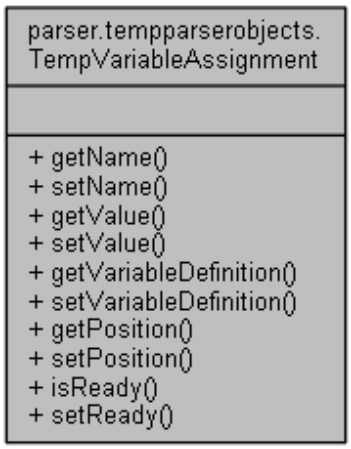
value	is the value.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/TempPackageRule.java

parser.tempparserobjects.TempVariableAssignment Klassenreferenz

parser.tempparserobjects.TempVariableAssignmentZusammengehörigkeiten von parser.tempparserobjects.TempVariableAssignment:



Öffentliche Methoden

- String getName ()
- void setName (final String name)
- String getValue ()
- void setValue (final String value)
- AbstractVariableDefinition getVariableDefinition ()
- void setVariableDefinition (final AbstractVariableDefinition varDef)
- Position getPosition ()
- void setPosition (final Position position)
- Boolean isReady ()
- void setReady (final Boolean ready)

Ausführliche Beschreibung

Autor:
Sandra

Dokumentation der Elementfunktionen

String parser.tempparserobjects.TempVariableAssignment.getName ()

Rückgabe:
name

Position parser.tempparserobjects.TempVariableAssignment.getPosition ()

Rückgabe:
return

String parser.tempparserobjects.TempVariableAssignment.getValue ()

Rückgabe:
value

AbstractVariableDefinition parser.tempparserobjects.TempVariableAssignment.getVariableDefinition ()

Rückgabe:
variableDefinition

Boolean parser.tempparserobjects.TempVariableAssignment.isReady ()

Rückgabe:
the ready

void parser.tempparserobjects.TempVariableAssignment.setName (final String *name*)

Parameter:

name	name

void parser.tempparserobjects.TempVariableAssignment.setPosition (final Position *position*)

Parameter:

position	the position to set

void parser.tempparserobjects.TempVariableAssignment.setReady (final Boolean *ready*)

Parameter:

ready	the ready to set

void parser.tempparserobjects.TempVariableAssignment.setValue (final String *value*)

Parameter:

value	value

void parser.tempparserobjects.TempVariableAssignment.setVariableDefinition (final AbstractVariableDefinition *varDef*)

Parameter:

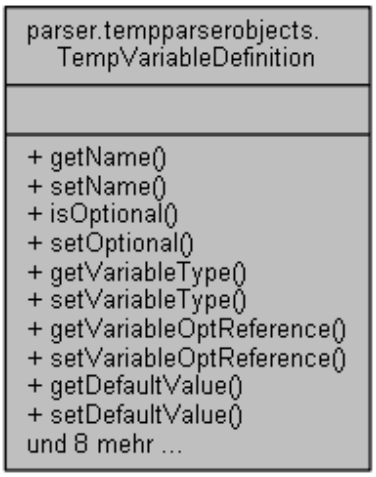
varDef	variableDefinition

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/tempparserobjects/TempVariableAssignment.java

parser.tempparserobjects.TempVariableDefinition Klassenreferenz

parser.tempparserobjects.TempVariableDefinitionZusammengehörigkeiten von parser.tempparserobjects.TempVariableDefinition:



Öffentliche Methoden

- String getName ()
- void setName (final String name)
- Boolean isOptional ()
- void setOptional (final Boolean optional)
- AbstractVariableType getVariableType ()
- void setVariableType (final AbstractVariableType variableType)
- VariableOptReference getVariableOptReference ()
- void setVariableOptReference (final VariableOptReference varOptRef)
- DefaultValue getDefaultValue ()
- void setDefaultValue (final DefaultValue defaultValue)
- AbstractVariableDefinition getVariableN ()
- void setVariableN (final AbstractVariableDefinition variableN)
- Position getPosition ()
- void setPosition (final Position position)
- Boolean isReady ()
- void setReady (final Boolean ready)
- Boolean isArrayElement ()
- void setArrayElement (final Boolean arrayElement)

Ausführliche Beschreibung

Autor:

Sandra

Dokumentation der Elementfunktionen

DefaultValue parser.tempparserobjects.TempVariableDefinition.getDefaultValue ()

Rückgabe:
defaultValue

String parser.tempparserobjects.TempVariableDefinition.getName ()

Rückgabe:
name

Position parser.tempparserobjects.TempVariableDefinition.getPosition ()

Rückgabe:
the row

AbstractVariableDefinition parser.tempparserobjects.TempVariableDefinition.getVariableN ()

Rückgabe:
abstractVariableDefinition

VariableOptReference parser.tempparserobjects.TempVariableDefinition.getVariableOptReference ()

Rückgabe:
variableOptReference

AbstractVariableType parser.tempparserobjects.TempVariableDefinition.getVariableType ()

Rückgabe:
variableType

Boolean parser.tempparserobjects.TempVariableDefinition.isArrayElement ()

Rückgabe:
the arrayElement

Boolean parser.tempparserobjects.TempVariableDefinition.isOptional ()

Rückgabe:
isOptional

Boolean parser.tempparserobjects.TempVariableDefinition.isReady ()

Rückgabe:
the ready

void parser.tempparserobjects.TempVariableDefinition.setArrayElement (final Boolean *arrayElement*)

Parameter:

arrayElement	the arrayElement to set

void parser.tempparserobjects.TempVariableDefinition.setDefaultValue (final DefaultValue *defaultValue*)

Parameter:

defaultValue	defaultValue

void parser.tempparserobjects.TempVariableDefinition.setName (final String *name*)

Parameter:

name	name

void parser.tempparserobjects.TempVariableDefinition.setOptional (final Boolean *optional*)

Parameter:

optional	isOptional

void parser.tempparserobjects.TempVariableDefinition.setPosition (final Position *position*)

Parameter:

position	the position to set

void parser.tempparserobjects.TempVariableDefinition.setReady (final Boolean *ready*)

Parameter:

ready	the ready to set

void parser.tempparserobjects.TempVariableDefinition.setVariableN (final AbstractVariableDefinition *variableN*)

Parameter:

variableN	abstractVariableDefinition

void parser.tempparserobjects.TempVariableDefinition.setVariableOptReference (final VariableOptReference *varOptRef*)

Parameter:

varOptRef	varOptRef

void parser.tempparserobjects.TempVariableDefinition.setVariableType (final AbstractVariableType *variableType*)

Parameter:

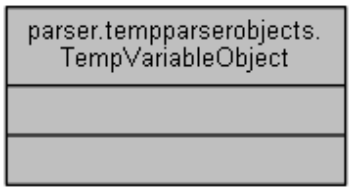
variableType	variableType

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/tempparserobjects/TempVariableDefinition.java

parser.tempparserobjects.TempVariableObject Schnittstellenreferenz

parser.tempparserobjects.TempVariableObjectZusammengehörigkeiten von parser.tempparserobjects.TempVariableObject:



Ausführliche Beschreibung

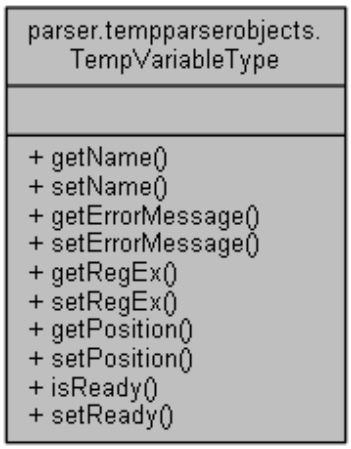
Autor:
Sandra

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/tempparserobjects/TempVariableObject.java

parser.tempparserobjects.TempVariableType Klassenreferenz

parser.tempparserobjects.TempVariableTypeZusammengehörigkeiten von parser.tempparserobjects.TempVariableType:



Öffentliche Methoden

- String getName ()
- void setName (final String name)
- String getErrorMessage ()
- void setErrorMessage (final String errorMessage)
- RegEx getRegEx ()
- void setRegEx (final RegEx regEx)
- Position getPosition ()
- void setPosition (final Position position)
- Boolean isReady ()
- void setReady (final Boolean ready)

Ausführliche Beschreibung

Autor:
Sandra

Dokumentation der Elementfunktionen

String parser.tempparserobjects.TempVariableType.getErrorMessage ()

Rückgabe:
errorMessage

String parser.tempparserobjects.TempVariableType.getName ()

Rückgabe:
name

Position parser.tempparserobjects.TempVariableType.getPosition ()

Rückgabe:
the row

RegEx parser.tempparserobjects.TempVariableType.getRegEx ()

Rückgabe:
parts

Boolean parser.tempparserobjects.TempVariableType.isReady ()

Rückgabe:
the ready

void parser.tempparserobjects.TempVariableType.setErrorMessage (final String *errorMessage*)

Parameter:

errorMessage	errormessage

void parser.tempparserobjects.TempVariableType.setName (final String *name*)

Parameter:

name	name

void parser.tempparserobjects.TempVariableType.setPosition (final Position *position*)

Parameter:

position	the position to set

void parser.tempparserobjects.TempVariableType.setReady (final Boolean *ready*)

Parameter:

ready	the ready to set

void parser.tempparserobjects.TempVariableType.setRegEx (final RegEx *regEx*)

stores a RegEx while input is a Lists of RexExParts.

Parameter:

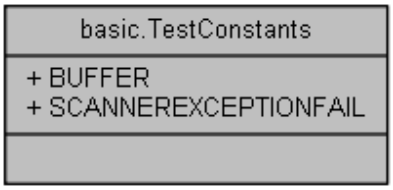
regEx	regEx

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/tempparserobjects/TempVariableType.java

basic.TestConstants Klassenreferenz

basic.TestConstantsZusammengehörigkeiten von basic.TestConstants:



Statische öffentliche Attribute

- static final Integer BUFFER = 50
- static final String SCANNEREXCEPTIONFAIL

Ausführliche Beschreibung

This class represents the constants of the scanner.

Autor:

HFW410

Dokumentation der Datenelemente

final Integer basic.TestConstants.BUFFER = 50`[static]`

The size of the buffer.

final String basic.TestConstants.SCANNEREXCEPTIONFAIL`[static]`

```
Initialisierung:=
            "Es wurde eine ScannerException erwartet, aber nicht ausgeloest"
```

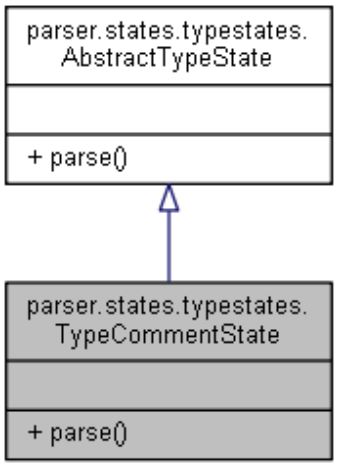
The String for a failed scanner exception.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

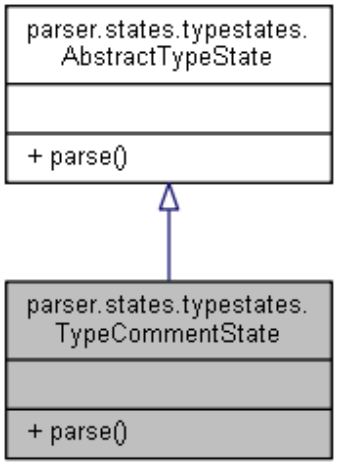
- workspace/ipfli4l13/src/basic/TestConstants.java

parser.states.typestates.TypeCommentState Klassenreferenz

parser.states.typestates.TypeCommentStateKlassendiagramm für parser.states.typestates.TypeCommentState:



Zusammengehörigkeiten von parser.states.typestates.TypeCommentState:



Öffentliche Methoden

- void parse (final TempVariableType tempVariableType, final Buffer< AbstractSymbol > buffer, final TypeFileParser tFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.typestates.TypeCommentState.parse (final TempVariableType *tempVariableType*, final Buffer< AbstractSymbol > *buffer*, final TypeFileParser *tFP*) throws InterruptedException, ParserException

Parameter:

tempVariableType	tempVariableType
buffer	buffer
tFP	typeFileParser

Rückgabe:

Ausnahmebehandlung:

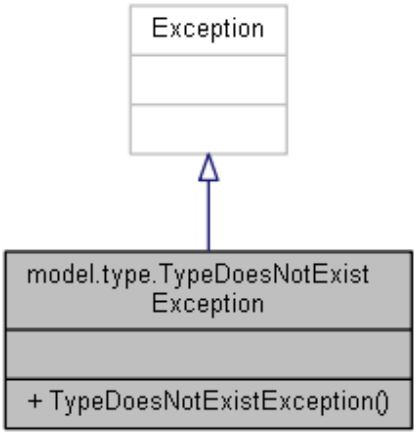
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

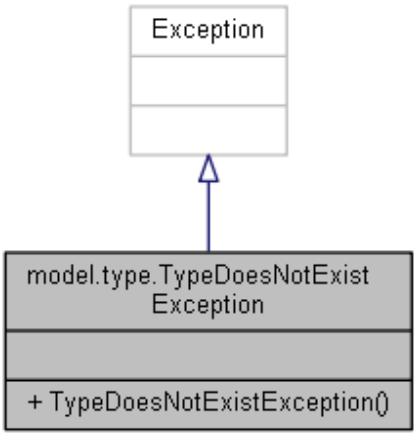
- workspace/ipfli4l13/src/parser/states/typestates/TypeCommentState.java

model.type.TypeDoesNotExistException Klassenreferenz

model.type.TypeDoesNotExistExceptionKlassendiagramm für model.type.TypeDoesNotExistException:



Zusammengehörigkeiten von model.type.TypeDoesNotExistException:



Öffentliche Methoden

- TypeDoesNotExistException (final String errorMessage)

Ausführliche Beschreibung

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.TypeDoesNotExistException.TypeDoesNotExistException (final String *errorMessage*)

Constructor, just calls super(errorMesage).

Parameter:

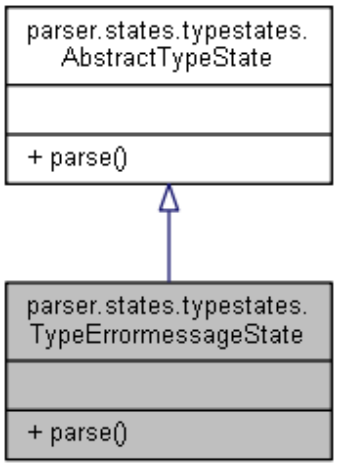
errorMessage	The message for the exception.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

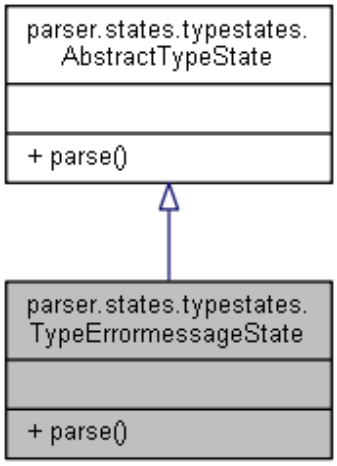
- workspace/ipfli4l13/src/model/type/TypeDoesNotExistException.java

parser.states.typestates.TypeErrormessageState Klassenreferenz

parser.states.typestates.TypeErrormessageStateKlassendiagramm für parser.states.typestates.TypeErrormessageState:



Zusammengehörigkeiten von parser.states.typestates.TypeErrormessageState:



Öffentliche Methoden

- void parse (final TempVariableType tVT, final Buffer< AbstractSymbol > buffer, final TypeFileParser tFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.typestates.TypeErrormessageState.parse (final TempVariableType *tVT*, final Buffer< AbstractSymbol > *buffer*, final TypeFileParser *tFP*) throws InterruptedException, ParserException

Parameter:

tVT	tempVariableType
buffer	buffer
tFP	typeFileParser

Rückgabe:

Ausnahmebehandlung:

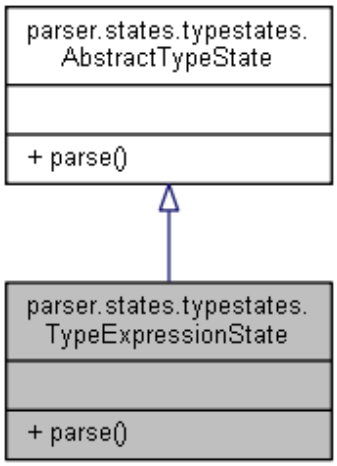
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

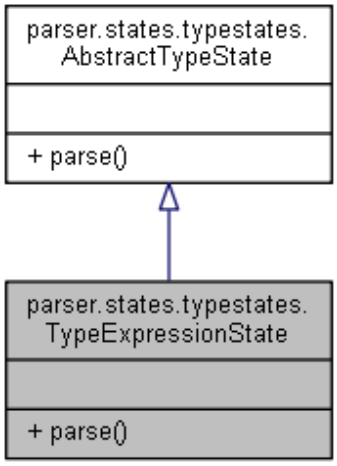
- workspace/ipfli4l13/src/parser/states/typestates/TypeErrormessageState.java

parser.states.typestates.TypeExpressionState Klassenreferenz

parser.states.typestates.TypeExpressionStateKlassendiagramm für parser.states.typestates.TypeExpressionState:



Zusammengehörigkeiten von parser.states.typestates.TypeExpressionState:



Öffentliche Methoden

- void parse (final TempVariableType tVT, final Buffer< AbstractSymbol > buffer, final TypeFileParser tFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.typestates.TypeExpressionState.parse (final TempVariableType *tVT*, final Buffer< AbstractSymbol > *buffer*, final TypeFileParser *tFP*) throws InterruptedException, ParserException

Parameter:

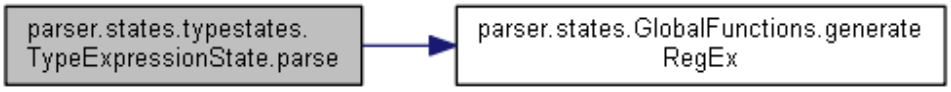
tVT	tempVariableType
buffer	buffer
tFP	typeFileParser

Rückgabe:

Ausnahmebehandlung:

InterruptedException	InterruptedException
ParserException	ParserException

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

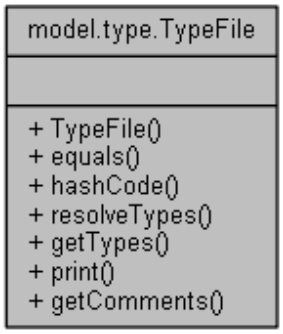


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/parser/states/typestates/TypeExpressionState.java

model.type.TypeFile Klassenreferenz

model.type.TypeFileZusammengehörigkeiten von model.type.TypeFile:



Öffentliche Methoden

- TypeFile (final Map< String, NamedVariableType > types2, final List< Comment > comments)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- void resolveTypes () throws TypeDoesNotExistException
- java.util.Map< String,
- NamedVariableType > getTypes ()
- String print (final String pathToPrint) throws printer.PrinterException
- java.util.List< Comment > getComments ()

Ausführliche Beschreibung

TypeFile represents the "\check\<package>.reg" files.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.TypeFile.TypeFile (final Map< String, NamedVariableType > *types2*, final List< Comment > *comments*)

Constructor, just sets fields. No side effects.

Parameter:

types2	the types to set.
comments	The comments to set.

Dokumentation der Elementfunktionen

java.util.List<Comment> model.type.TypeFile.getComments ()

Rückgabe:

the comments

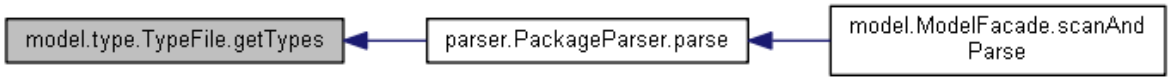
java.util.Map<String, NamedVariableType> model.type.TypeFile.getTypes ()

Returns the field type. No side effects.

Rückgabe:

this.types

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



String model.type.TypeFile.print (final String *pathToPrint*) throws printer.PrinterException

Prints an TypeFile containing the VariableTypes and Comments.

Parameter:

pathToPrint	The path where to print the file.

Ausnahmebehandlung:

printer.PrinterException	If a IOException is thrown.
printer.PrinterException	

Rückgabe:

Returns a string representing the printed lines (for testing).

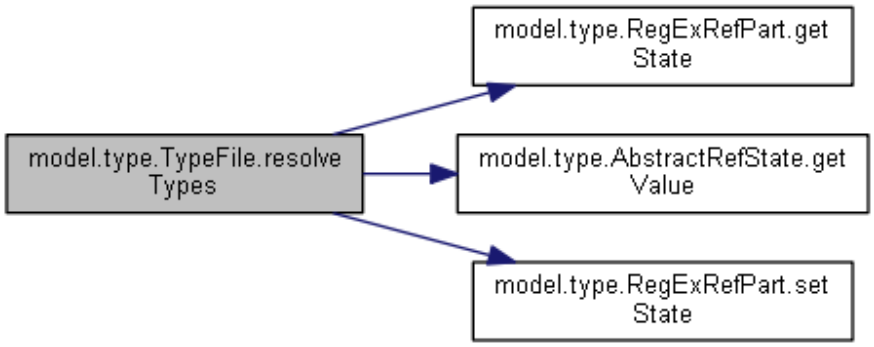
void model.type.TypeFile.resolveTypes () throws TypeDoesNotExistException

Resolves the unresolved RegExparts of the types.

Ausnahmebehandlung:

TypeDoesNotExistException	if a part is unresolvable cause the type to resolve doesnt exit.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

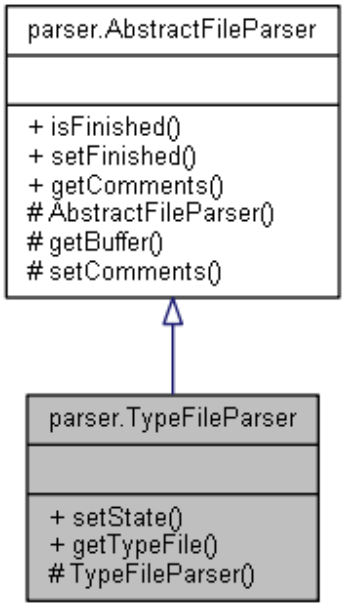


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

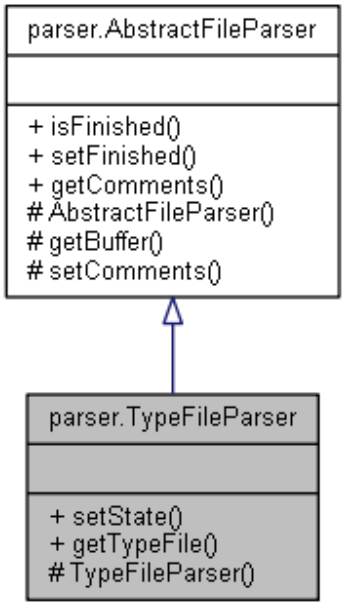
- workspace/ipfli4l13/src/model/type/TypeFile.java

parser.TypeFileParser Klassenreferenz

parser.TypeFileParserKlassendiagramm für parser.TypeFileParser:



Zusammengehörigkeiten von parser.TypeFileParser:



Öffentliche Methoden

- void setState (final AbstractTypeState state)

- TypeFile getTypeFile ()

Geschützte Methoden

- TypeFileParser (final Buffer< AbstractSymbol > buffer) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
hfw410wi - Mark Wittig

Beschreibung der Konstruktoren und Destruktoren

parser.TypeFileParser.TypeFileParser (final Buffer< AbstractSymbol > *buffer*) throws InterruptedException, ParserException `[protected]`

The TypeFileParser will automatically generate a TypeFile after instantiation which can be obtained via getTypeFile().

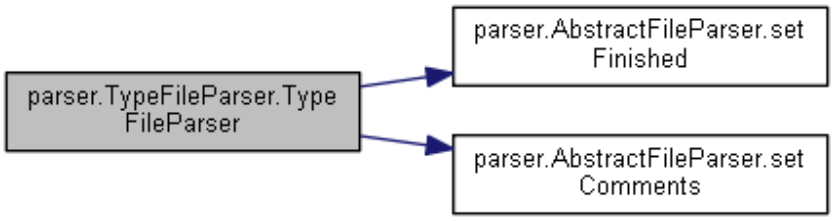
Parameter:

buffer	Buffer<Symbol>

Ausnahmebehandlung:

ParserException	Parser Exception
InterruptedException	InterruptedException

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



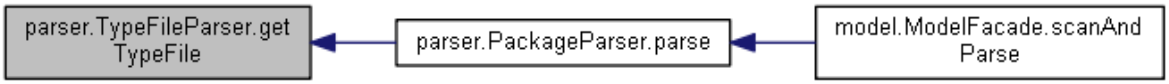
Dokumentation der Elementfunktionen

TypeFile parser.TypeFileParser.getTypeFile ()

Returns the desired Type File which is automatically created right after instantiation.

Rückgabe:
the typeFile

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



void parser.TypeFileParser.setState (final AbstractTypeState *state*)

Parameter:

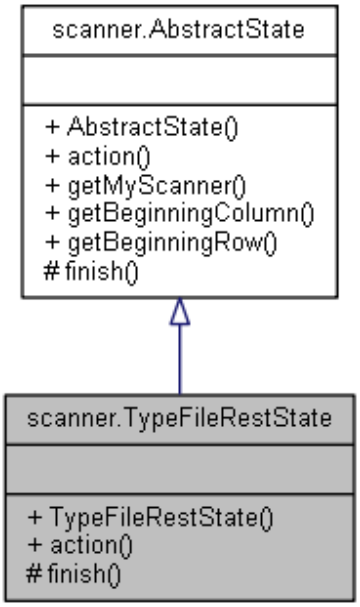
state	the state to set

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

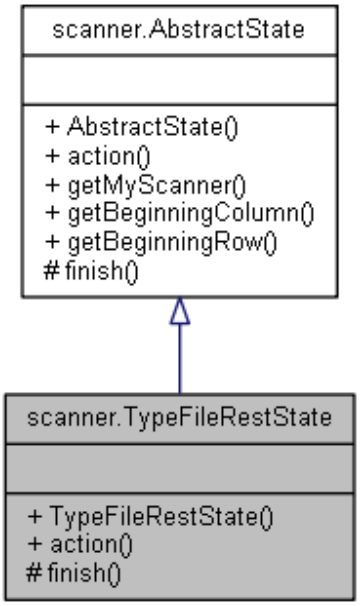
- workspace/ipfli4l13/src/parser/TypeFileParser.java

scanner.TypeFileRestState Klassenreferenz

scanner.TypeFileRestStateKlassendiagramm für scanner.TypeFileRestState:



Zusammengehörigkeiten von scanner.TypeFileRestState:



Öffentliche Methoden

- TypeFileRestState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)

- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath) throws InterruptedException

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath)

Ausführliche Beschreibung

This class represents the RestState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.TypeFileRestState.TypeFileRestState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

This constructor initialize the RestState.

Parameter:

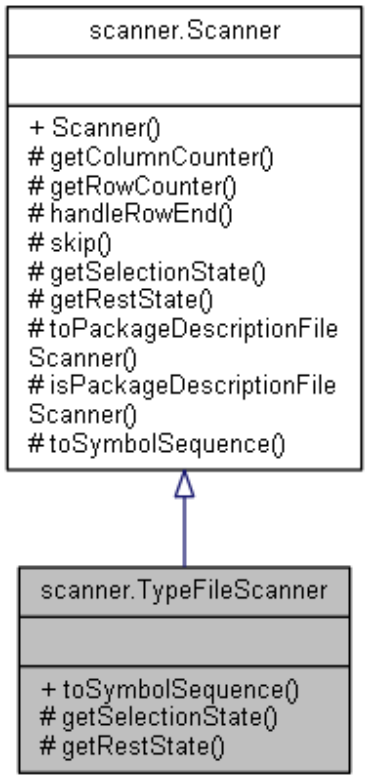
myScanner	is the used Scanner.
beginningColumn	is the Begin Column of the Symbol
beginningRow	is the beginning row of the symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

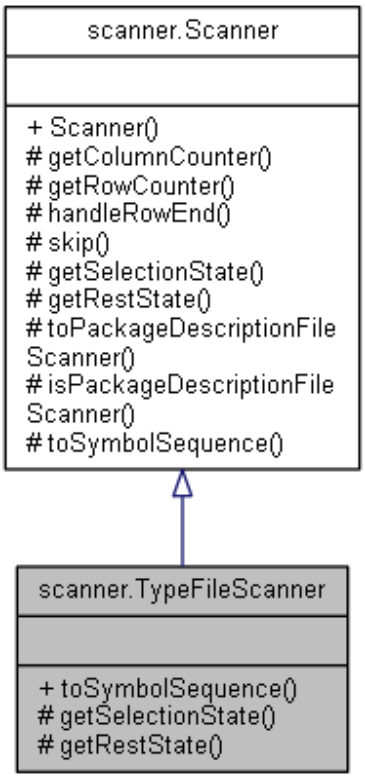
- workspace/ipfli4l13/src/scanner/TypeFileRestState.java

scanner.TypeFileScanner Klassenreferenz

scanner.TypeFileScannerKlassendiagramm für scanner.TypeFileScanner:



Zusammengehörigkeiten von scanner.TypeFileScanner:



Öffentliche, statische Methoden

- static void toSymbolSequence (final InputStream input, final String dataPath, final Buffer< AbstractSymbol > buffer) throws ScannerException, InterruptedException, IOException

Geschützte Methoden

- AbstractState getSelectionState ()
- AbstractState **getRestState** (final Integer beginningRow, final Integer beginningColumn)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represent the scanner for a TypeFile.

Autor:

HFW410

Dokumentation der Elementfunktionen

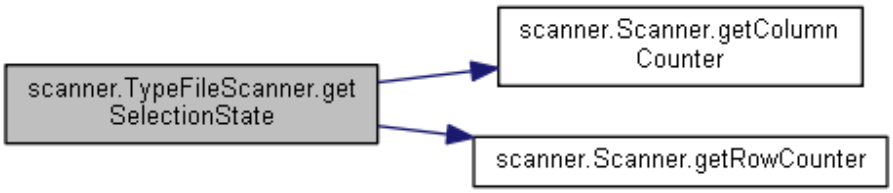
AbstractState scanner.TypeFileScanner.getSelectionState ()[protected], [virtual]

Rückgabe:

the SelectionState of the special scanner.

Implementiert scanner.Scanner.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



static void scanner.TypeFileScanner.toSymbolSequence (final InputStream *input*, final String *dataPath*, final Buffer< AbstractSymbol > *buffer*) throws ScannerException, InterruptedException, IOException `[static]`

creates a scanner and starts him.

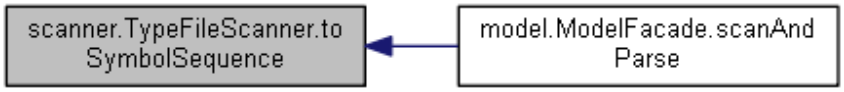
Parameter:

input	is the input stream.
dataPath	is the path of the file.
buffer	if the buffer for the symbols.

Ausnahmebehandlung:

IOException	from the Input Stream.
InterruptedException	from the buffer.
ScannerException	if there is any problem during the scan.

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

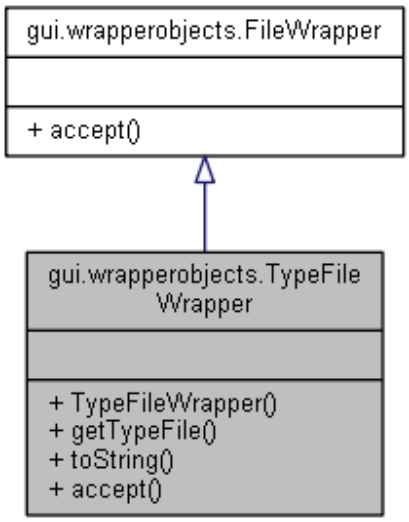


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

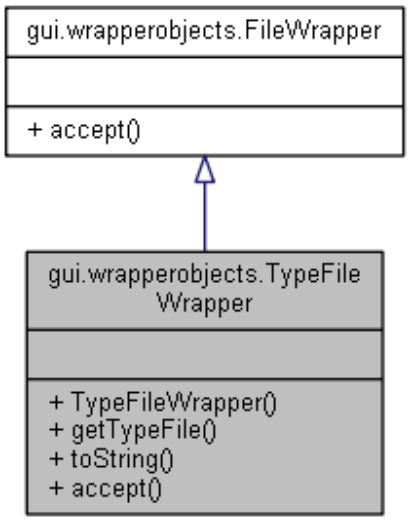
- workspace/ipfli4l13/src/scanner/TypeFileScanner.java

gui.wrapperobjects.TypeFileWrapper Klassenreferenz

gui.wrapperobjects.TypeFileWrapperKlassendiagramm für gui.wrapperobjects.TypeFileWrapper:



Zusammengehörigkeiten von gui.wrapperobjects.TypeFileWrapper:



Öffentliche Methoden

- TypeFileWrapper (final TypeFile typeFile)
- TypeFile getTypeFile ()
- String **toString** ()
- void accept (final FileWrapperVisitor vis)

Ausführliche Beschreibung

A Wrapperclass for TypeFile.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.wrapperobjects.TypeFileWrapper.TypeFileWrapper (final TypeFile *typeFile*)

Constructor for TypeFileWrapper.

Parameter:

typeFile	The typeFile to wrap.

Dokumentation der Elementfunktionen

void gui.wrapperobjects.TypeFileWrapper.accept (final FileWrapperVisitor *vis*)

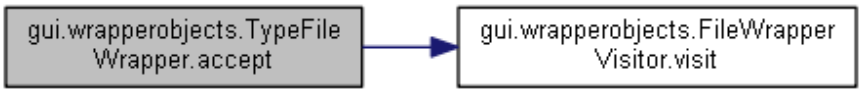
accept-method for visitor pattern.

Parameter:

vis	the visitor.

Implementiert gui.wrapperobjects.FileWrapper.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



TypeFile gui.wrapperobjects.TypeFileWrapper.getTypeFile ()

Rückgabe:

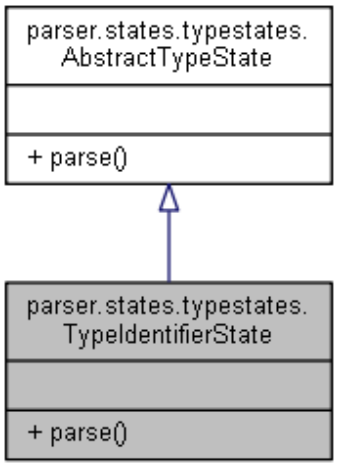
the typeFile

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

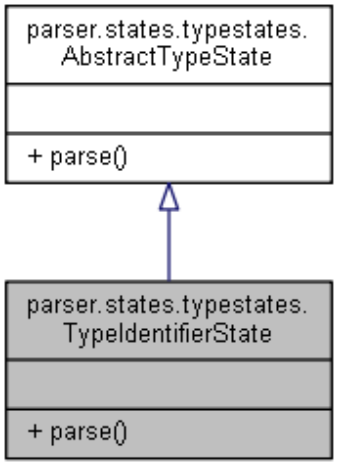
- workspace/ipfli4l13/src/gui/wrapperobjects/TypeFileWrapper.java

parser.states.typestates.TypeIdentifierState Klassenreferenz

parser.states.typestates.TypeIdentifierStateKlassendiagramm für parser.states.typestates.TypeIdentifierState:



Zusammengehörigkeiten von parser.states.typestates.TypeIdentifierState:



Öffentliche Methoden

- void parse (final TempVariableType tVT, final Buffer< AbstractSymbol > buffer, final TypeFileParser tFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.typestates.TypeIdentifierState.parse (final TempVariableType *tVT*, final Buffer< AbstractSymbol > *buffer*, final TypeFileParser *tFP*) throws InterruptedException, ParserException

Parameter:

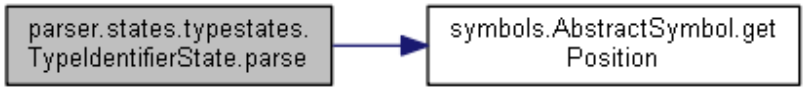
tVT	tempVariableType
buffer	buffer
tFP	typeFileParser

Rückgabe:

Ausnahmebehandlung:

InterruptedException	InterruptedException
ParserException	ParserException

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

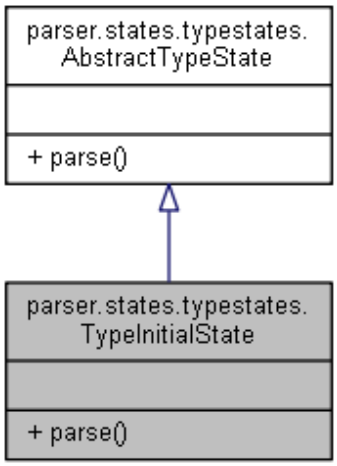


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

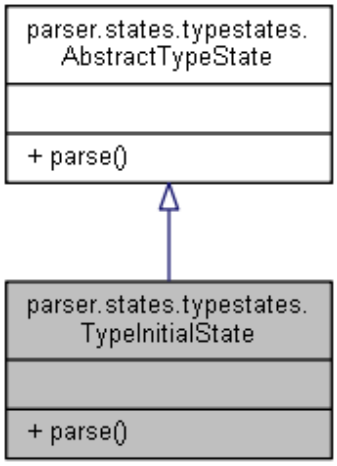
- workspace/ipfli4l13/src/parser/states/typestates/TypeIdentifierState.java

parser.states.typestates.TypeInitialState Klassenreferenz

parser.states.typestates.TypeInitialStateKlassendiagramm für parser.states.typestates.TypeInitialState:



Zusammengehörigkeiten von parser.states.typestates.TypeInitialState:



Öffentliche Methoden

- void parse (final TempVariableType tVT, final Buffer< AbstractSymbol > buffer, final TypeFileParser tFP) throws InterruptedException, ParserException

---

Ausführliche Beschreibung

Autor:
Muri

Dokumentation der Elementfunktionen

void parser.states.typestates.TypeInitialState.parse (final TempVariableType *tVT*, final Buffer< AbstractSymbol > *buffer*, final TypeFileParser *tFP*) throws InterruptedException, ParserException

Parameter:

tVT	tempVariableType
buffer	buffer
tFP	typeFileParser

Rückgabe:

Ausnahmebehandlung:

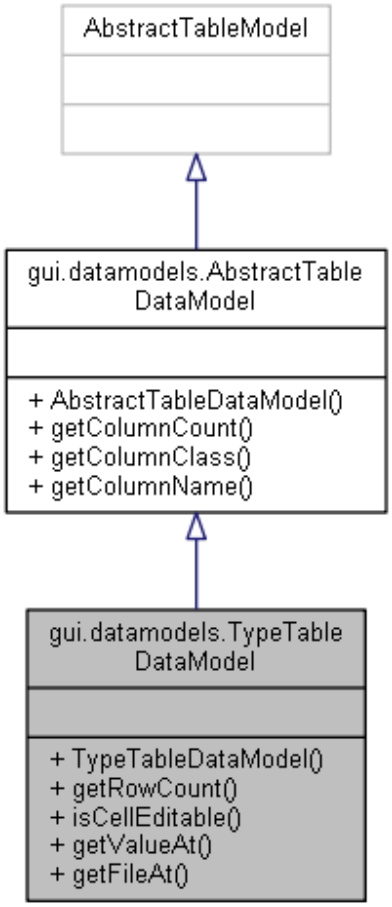
InterruptedException	InterruptedException
ParserException	ParserException

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

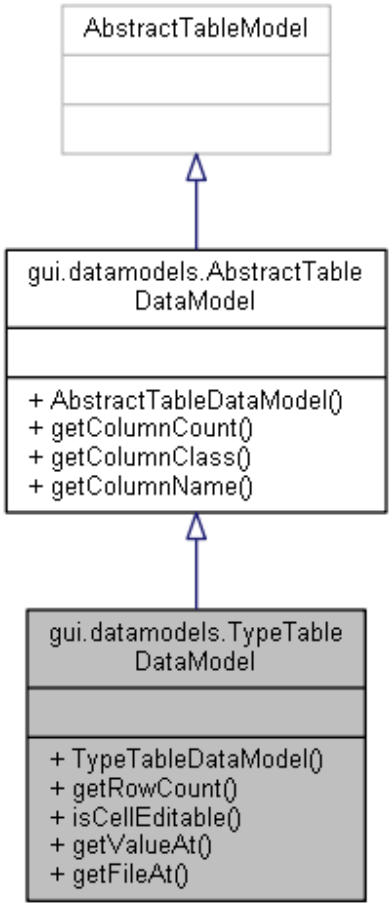
- workspace/ipfli4l13/src/parser/states/typestates/TypeInitialState.java

gui.datamodels.TypeTableDataModel Klassenreferenz

gui.datamodels.TypeTableDataModelKlassendiagramm für gui.datamodels.TypeTableDataModel:



Zusammengehörigkeiten von gui.datamodels.TypeTableDataModel:



Öffentliche Methoden

- TypeTableDataModel (final List< NamedVariableType > data)
- int **getRowCount** ()
- boolean **isCellEditable** (final int r, final int c)
- Object **getValueAt** (final int r, final int c)
- NamedVariableType getFileAt (final int r)

Ausführliche Beschreibung

tableModel for typefiles.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.datamodels.TypeTableDataModel.TypeTableDataModel (final List< NamedVariableType > *data*)

Constructor, wraps the given data into a list of wraped data and sorts the list by the positions of entries.

Parameter:

data	the data to initialize the table.

Dokumentation der Elementfunktionen

NamedVariableType gui.datamodels.TypeTableDataModel.getFileAt (final int *r*)

Returns the VariableAssignment of the row r.

Parameter:

r	The row r.

Rückgabe:

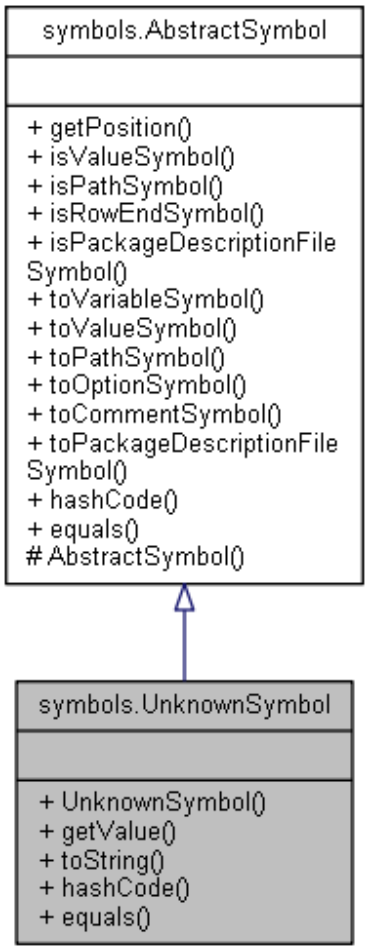
the variable assignment.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

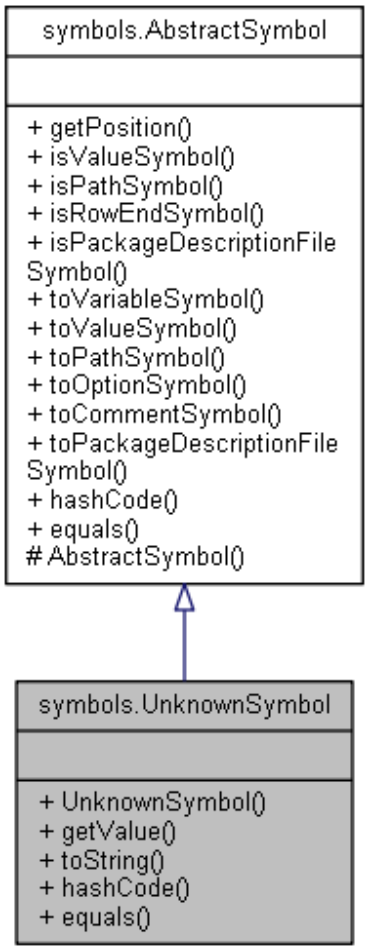
- workspace/ipfli4l13/src/gui/datamodels/TypeTableDataModel.java

symbols.UnknownSymbol Klassenreferenz

symbols.UnknownSymbolKlassendiagramm für symbols.UnknownSymbol:



Zusammengehörigkeiten von symbols.UnknownSymbol:



Öffentliche Methoden

- UnknownSymbol (final String value, final Position position)
- String getValue ()
- String **toString** ()
- int **hashCode** ()
- boolean **equals** (final Object obj)

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents a Unkown Symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.UnknownSymbol.UnknownSymbol (final String *value*, final Position *position*)

This is only the constructor of the UnkownSymbol. It initialize the Unknown Symbol and set

Parameter:

value	= this.value.
position	is the row and column of the data in the stream.

Dokumentation der Elementfunktionen

String symbols.UnknownSymbol.getValue ()

This attribute

Rückgabe:

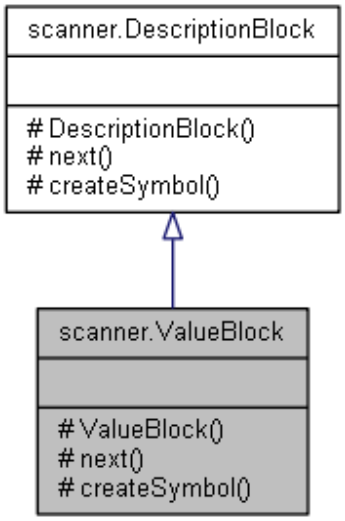
the value of the unknown symbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

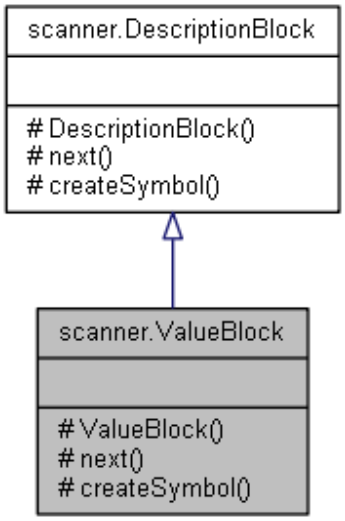
- workspace/ipfli4l13/src/symbols/UnknownSymbol.java

scanner.ValueBlock Klassenreferenz

scanner.ValueBlockKlassendiagramm für scanner.ValueBlock:



Zusammengehörigkeiten von scanner.ValueBlock:



Geschützte Methoden

- ValueBlock ()
- scanner.DescriptionBlock next ()
- AbstractSymbol createSymbol (final String collectedContent, final Position position)

Ausführliche Beschreibung

This class represents the second Block (the value block).

Autor:

Hendrik

Beschreibung der Konstruktoren und Destruktoren

scanner.ValueBlock.ValueBlock ()`[protected]`

constructor.

Dokumentation der Elementfunktionen

AbstractSymbol scanner.ValueBlock.createSymbol (final String *collectedContent*, final Position *position*)`[protected]`, `[virtual]`

Create the new Symbol.

Parameter:

collectedContent	collected Content which is used for the Symbol.
position	the position of the collected content.

Rückgabe:

the new Symbol

Implementiert scanner.DescriptionBlock.

scanner.DescriptionBlock scanner.ValueBlock.next ()`[protected]`, `[virtual]`

set the next State.

Rückgabe:

the new State

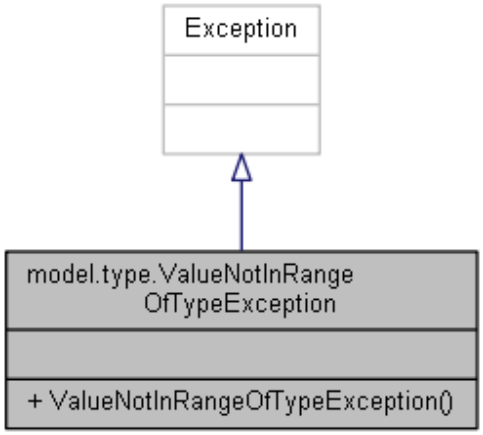
Implementiert scanner.DescriptionBlock.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

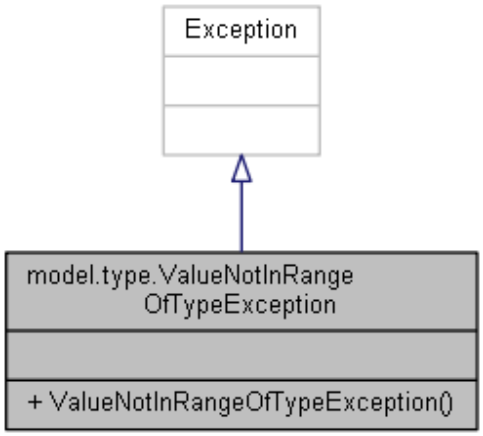
- workspace/ipfli4l13/src/scanner/ValueBlock.java

model.type.ValueNotInRangeOfTypeException Klassenreferenz

model.type.ValueNotInRangeOfTypeExceptionKlassendiagramm für model.type.ValueNotInRangeOfTypeException:



Zusammengehörigkeiten von model.type.ValueNotInRangeOfTypeException:



Öffentliche Methoden

- ValueNotInRangeOfTypeException (final String errorMessage)

Ausführliche Beschreibung

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.type.ValueNotInRangeOfTypeException.ValueNotInRangeOfTypeException (final String *errorMessage*)

Constructor, just calls super(errorMesage).

Parameter:

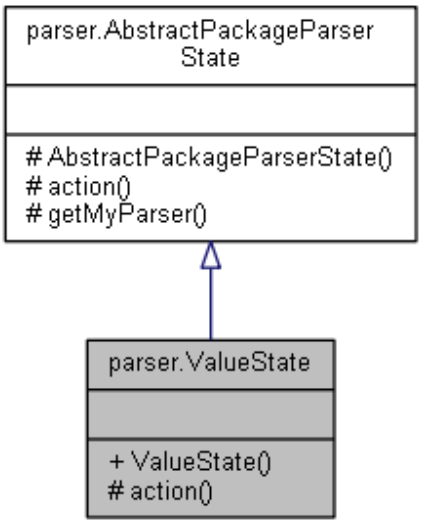
errorMessage	The message for the exception.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

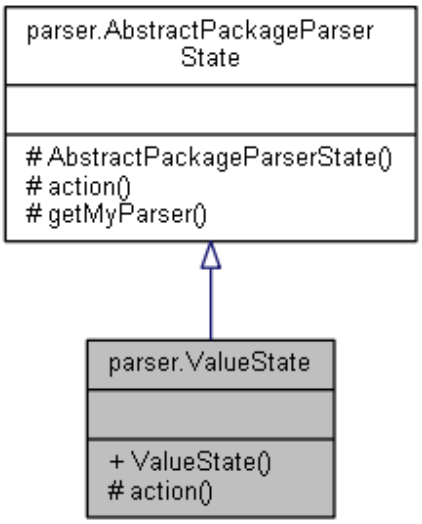
- workspace/ipfli4l13/src/model/type/ValueNotInRangeOfTypeException.java

parser.ValueState Klassenreferenz

parser.ValueStateKlassendiagramm für parser.ValueState:



Zusammengehörigkeiten von parser.ValueState:



Öffentliche Methoden

- ValueState (final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule)

Geschützte Methoden

- AbstractPackageParserState **action** (final PackageDescriptionFileSymbol symbol, final Map< String, AbstractVariableDefinition > variableDef) throws InterruptedException, ParserException

Ausführliche Beschreibung

This class represents the value State for the PackageDescriptionFileParser.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.ValueState.ValueState (final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*)

The constructor for the ValueState.

Parameter:

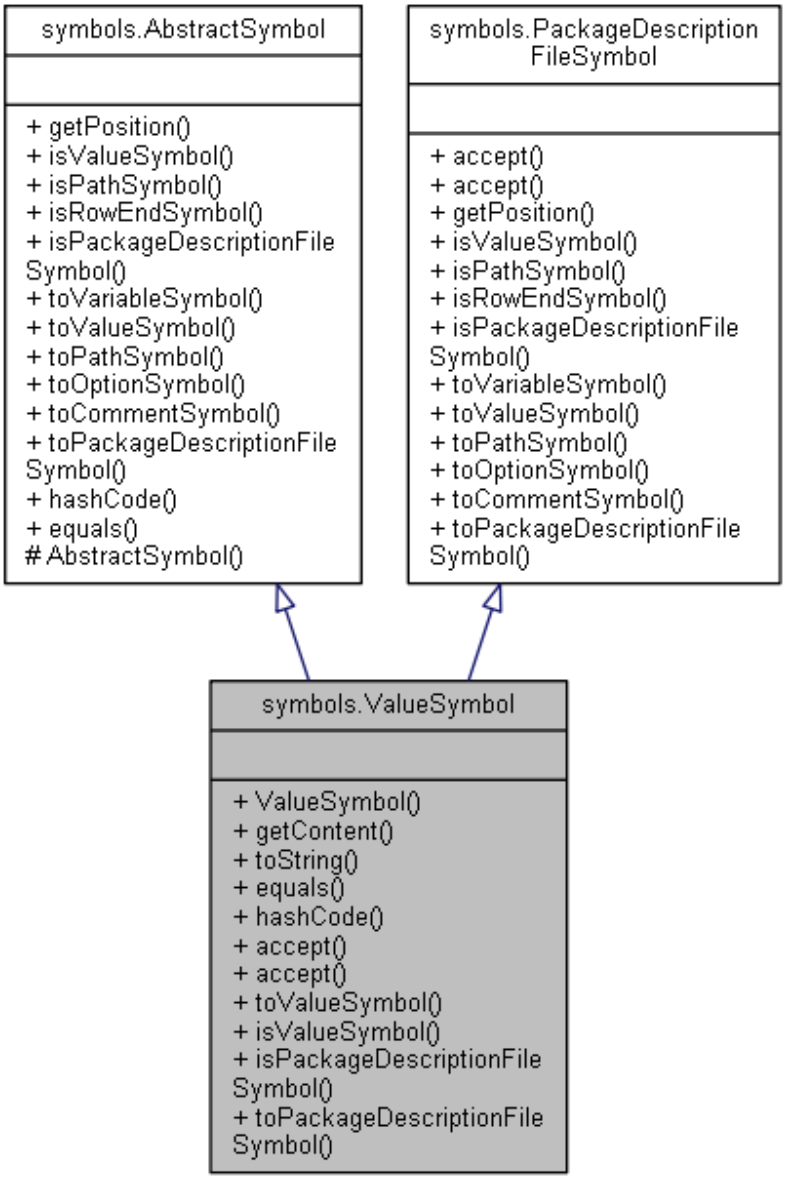
parser	for the Descriptionfiles
tempPackageRule	temp packageRule

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

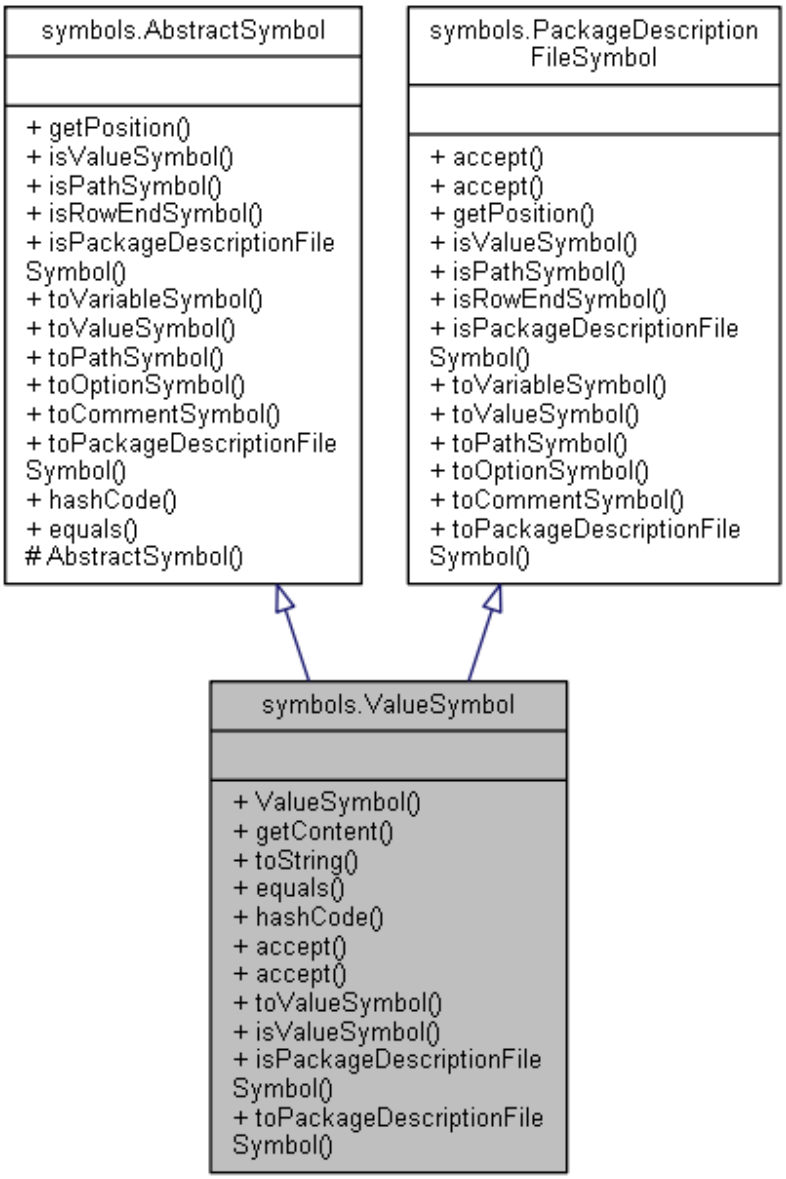
- workspace/ipfli4l13/src/parser/ValueState.java

symbols.ValueSymbol Klassenreferenz

symbols.ValueSymbolKlassendiagramm für symbols.ValueSymbol:



Zusammengehörigkeiten von symbols.ValueSymbol:



Öffentliche Methoden

- ValueSymbol (final String content, final Position position)
- String getContent ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitor visitor, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitorWithTempRule visitor, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- ValueSymbol toValueSymbol ()
- Boolean isValueSymbol ()

- Boolean isPackageDescriptionFileSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the value symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.ValueSymbol.ValueSymbol (final String *content*, final Position *position*)

This constructor instantiate the value symbol.

Parameter:

position	is the position of the data in the stream.
content	is the content of the ValueState.

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.ValueSymbol.accept (final PackageDescriptionSymbolVisitor *visitor*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

AbstractPackageParserState symbols.ValueSymbol.accept (final PackageDescriptionSymbolVisitorWithTempRule *visitor*, final

PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.
tempPackageRule	is a temporary package rule.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

String symbols.ValueSymbol.getContent ()

Rückgabe:

the content of the ValueSymbol.

Boolean symbols.ValueSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

Boolean symbols.ValueSymbol.isValueSymbol ()

Rückgabe:

true, if the Symbol is a value symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

PackageDescriptionFileSymbol symbols.ValueSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:

a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

ValueSymbol symbols.ValueSymbol.toValueSymbol ()

Rückgabe:

an error, if the symbol is not a value symbol

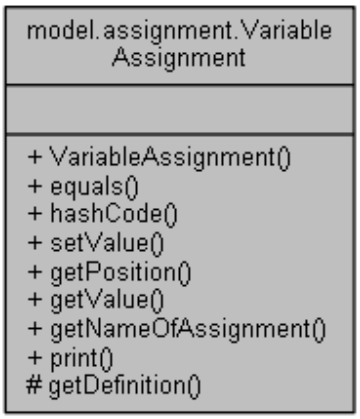
Implementiert symbols.PackageDescriptionFileSymbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/symbols/ValueSymbol.java

model.assignment.VariableAssignment Klassenreferenz

model.assignment.VariableAssignmentZusammengehörigkeiten von model.assignment.VariableAssignment:



Öffentliche Methoden

- VariableAssignment (final String nameOfAssignment, final model.definition.AbstractVariableDefinition definition, final String value, final Position position)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- void setValue (final String value) throws ValueNotInRangeOfTypeException, IllegalArgumentException, ReferenceIsUnresolvedException
- Position getPosition ()
- String getValue ()
- String getNameOfAssignment ()
- String print ()

Geschützte Methoden

- model.definition.AbstractVariableDefinition getDefinition ()

Ausführliche Beschreibung

VariableAssignment represents an entry in the "\config\<package>.txt" files.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.assignment.VariableAssignment.VariableAssignment (final String *nameOfAssignment*, final model.definition.AbstractVariableDefinition *definition*, final String *value*, final Position *position*)

Constructor, just sets the fields.

Parameter:

definition	VariableDefinition to set (field)
value	Value to set (field)
nameOfAssignment	The name to set.
position	The position to set.

Dokumentation der Elementfunktionen

[model.definition.AbstractVariableDefinition](#) model.assignment.VariableAssignment.getDefinition ()`[protected]`

Returns the field definition. No side effects.

Rückgabe:

this.definition

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

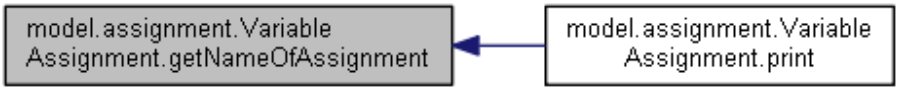
String model.assignment.VariableAssignment.getNameOfAssignment ()

Returns the field name. No side effects.

Rückgabe:

this.name

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



[Position](#) model.assignment.VariableAssignment.getPosition ()

Returns the field row. No side effects.

Rückgabe:

this.row

String model.assignment.VariableAssignment.getValue ()

Returns the field value. No side effects.

Rückgabe:

this.value

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

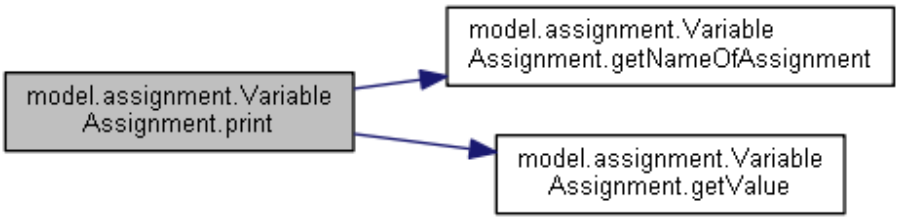
String model.assignment.VariableAssignment.print ()

Returns the line to print.

Rückgabe:

Returns a string value, representing a line to print.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



void model.assignment.VariableAssignment.setValue (final String *value*) throws ValueNotInRangeOfTypeException, IllegalArgumentException, ReferenceIsUnresolvedException

Try to set a new value to the variable (field). If the value is not in range of the given type of the variable, a ValueNotInRangeOfTypeException is thrown. checkValue(String name) is called.

Parameter:

value	The value to set

Ausnahmebehandlung:

ValueNotInRangeOfTypeException	The value is not in the range of the given type
IllegalArgumentException	
ReferenceIsUnresolvedException	A part of the regular expression is unresolved.
IllegalArgumen-	see java.lang.IllegalArgumentException

tException	
CycleInHierar-chyDetectedEx-ception	

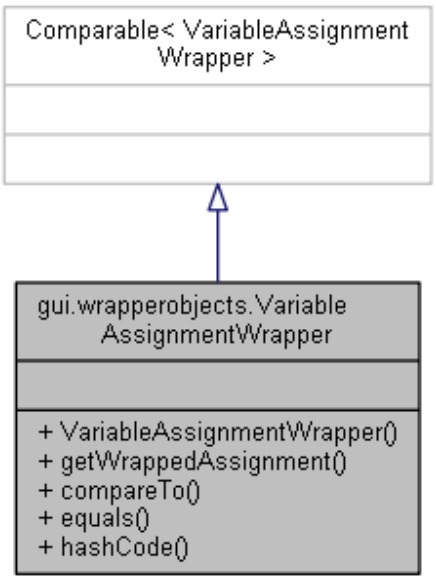
Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

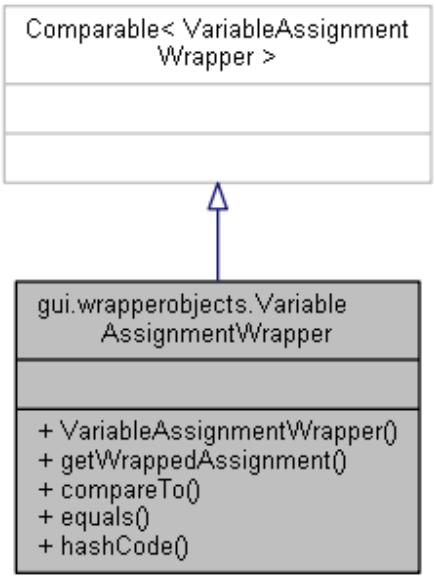
- workspace/ipfli4l13/src/model/assignment/VariableAssignment.java

gui.wrapperobjects.VariableAssignmentWrapper Klassenreferenz

gui.wrapperobjects.VariableAssignmentWrapperKlassendiagramm für gui.wrapperobjects.VariableAssignmentWrapper:



Zusammengehörigkeiten von gui.wrapperobjects.VariableAssignmentWrapper:



Öffentliche Methoden

- VariableAssignmentWrapper (final VariableAssignment toWrap)
- VariableAssignment getWrappedAssignment ()

- int **compareTo** (final VariableAssignmentWrapper o)
- boolean **equals** (final Object o)
- int **hashCode** ()

Ausführliche Beschreibung

This class wraps a VariableAssignment.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.wrapperobjects.VariableAssignmentWrapper.VariableAssignmentWrapper (final VariableAssignment *toWrap*)

Constructor, just sets fields. No side-effects.

Parameter:

toWrap	The assignment to wrap.

Dokumentation der Elementfunktionen

VariableAssignment gui.wrapperobjects.VariableAssignmentWrapper.getWrappedAssignment ()

Rückgabe:

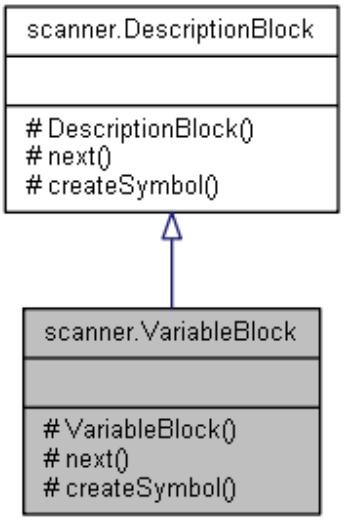
the wrappedAssignment

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

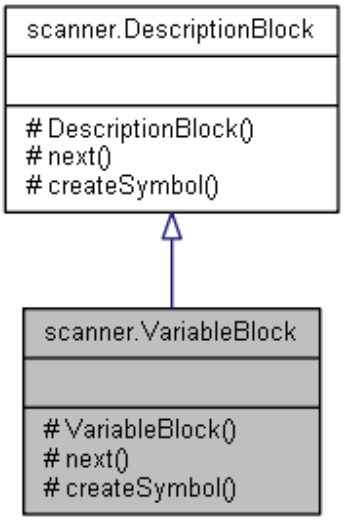
- workspace/ipfli4l13/src/gui/wrapperobjects/VariableAssignmentWrapper.java

scanner.VariableBlock Klassenreferenz

scanner.VariableBlockKlassendiagramm für scanner.VariableBlock:



Zusammengehörigkeiten von scanner.VariableBlock:



Geschützte Methoden

- VariableBlock ()
- DescriptionBlock next ()
- AbstractSymbol createSymbol (final String collectedContent, final Position position)

Ausführliche Beschreibung

This class represents the Variable Block in the PackageDescriptionFile.

Autor:

Hendrik

Beschreibung der Konstruktoren und Destruktoren

scanner.VariableBlock.VariableBlock ()`[protected]`

constructor.

Dokumentation der Elementfunktionen

AbstractSymbol scanner.VariableBlock.createSymbol (final String *collectedContent*, final Position *position*)`[protected]`, `[virtual]`

Create the new Symbol.

Parameter:

collectedContent	collected Content which is used for the Symbol.
position	the position of the collected content.

Rückgabe:

the new Symbol

Implementiert scanner.DescriptionBlock.

DescriptionBlock scanner.VariableBlock.next ()`[protected]`, `[virtual]`

set the next State.

Rückgabe:

the new State

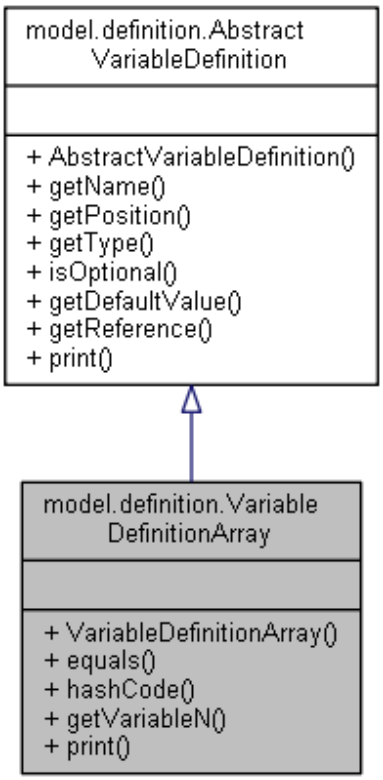
Implementiert scanner.DescriptionBlock.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

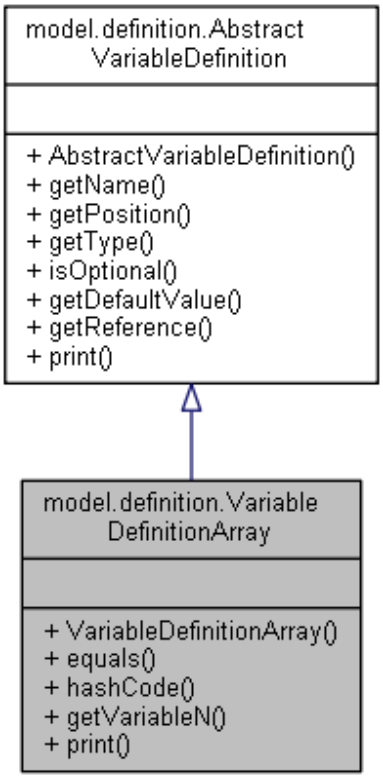
- workspace/ipfli4l13/src/scanner/VariableBlock.java

model.definition.VariableDefinitionArray Klassenreferenz

model.definition.VariableDefinitionArrayKlassendiagramm für model.definition.VariableDefinitionArray:



Zusammengehörigkeiten von model.definition.VariableDefinitionArray:



Öffentliche Methoden

- VariableDefinitionArray (final String name, final model.type.AbstractVariableType type, final boolean isOptional, final DefaultValue defaultValue, final VariableOptReference reference, final AbstractVariableDefinition variableN, final Position position)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- AbstractVariableDefinition getVariableN ()
- String print ()

Ausführliche Beschreibung

VariableDefinitionArray extends VariableDefinition and represents an ArrayVariable.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.definition.VariableDefinitionArray.VariableDefinitionArray (final String *name*, final model.type.AbstractVariableType *type*, final boolean

***isOptional*, final DefaultValue *defaultValue*, final VariableOptReference *reference*, final AbstractVariableDefinition *variableN*, final Position *position*)**

Constructor, just sets the fields. No side effects.

Parameter:

name	The name to set.
type	The type to set.
isOptional	Represents if an definition is optional.
defaultValue	The default value to set.
reference	Represents if a definition has an opt or not.
variableN	Reference to the "parent"-definition
position	the position to set.

Dokumentation der Elementfunktionen

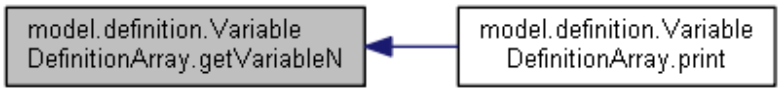
AbstractVariableDefinition model.definition.VariableDefinitionArray.getVariableN ()

Returns the field variableN. No side effects.

Rückgabe:

this.variableN

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



String model.definition.VariableDefinitionArray.print ()[virtual]

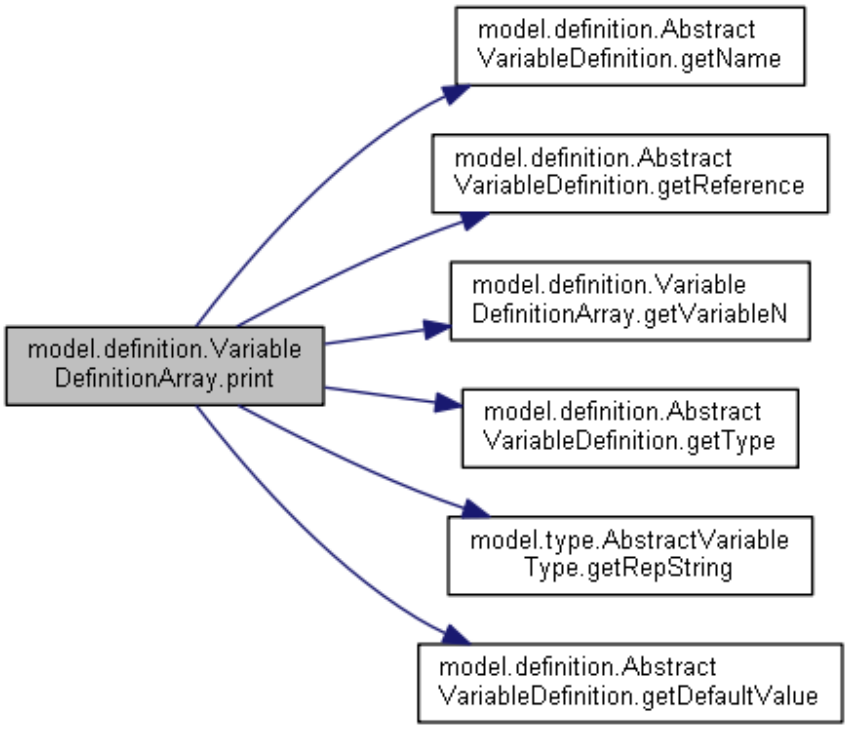
Returns the line to print.

Rückgabe:

Returns a string value, representing a line to print.

Implementiert model.definition.AbstractVariableDefinition.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

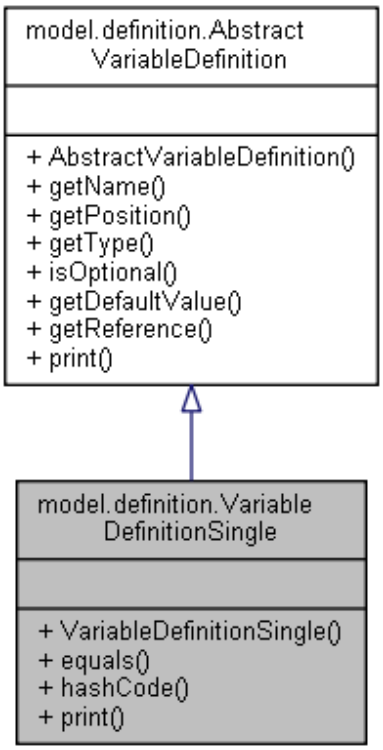


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

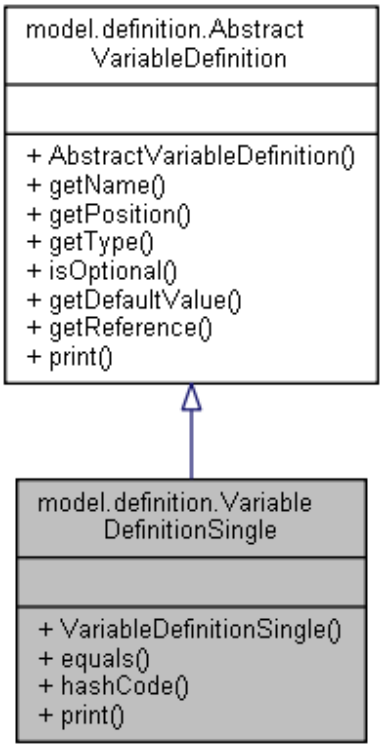
- workspace/ipfli4l13/src/model/definition/VariableDefinitionArray.java

model.definition.VariableDefinitionSingle Klassenreferenz

model.definition.VariableDefinitionSingleKlassendiagramm für model.definition.VariableDefinitionSingle:



Zusammengehörigkeiten von model.definition.VariableDefinitionSingle:



Öffentliche Methoden

- VariableDefinitionSingle (final String name, final AbstractVariableType type, final boolean isOptional, final DefaultValue defaultValue, final VariableOptReference reference, final Position position)
- boolean **equals** (final Object obj)
- int **hashCode** ()
- String print ()

Ausführliche Beschreibung

VariableDefinitionSingle extends VariableDefinition and represents a SingleVariable.

Autor:

HFW410RA - Philipp Rammos

Beschreibung der Konstruktoren und Destruktoren

model.definition.VariableDefinitionSingle.VariableDefinitionSingle (final String *name*, final AbstractVariableType *type*, final boolean *isOptional*, final DefaultValue *defaultValue*, final VariableOptReference *reference*, final Position *position*)

Constructor, just sets the fields.

Parameter:

name	The name to set.
type	The type to set.
isOptional	Represents if an definition is optional.
defaultValue	The default value to set.
reference	Represents if a definition has an opt or not.
position	the position to set.

Dokumentation der Elementfunktionen

String model.definition.VariableDefinitionSingle.print ()[virtual]

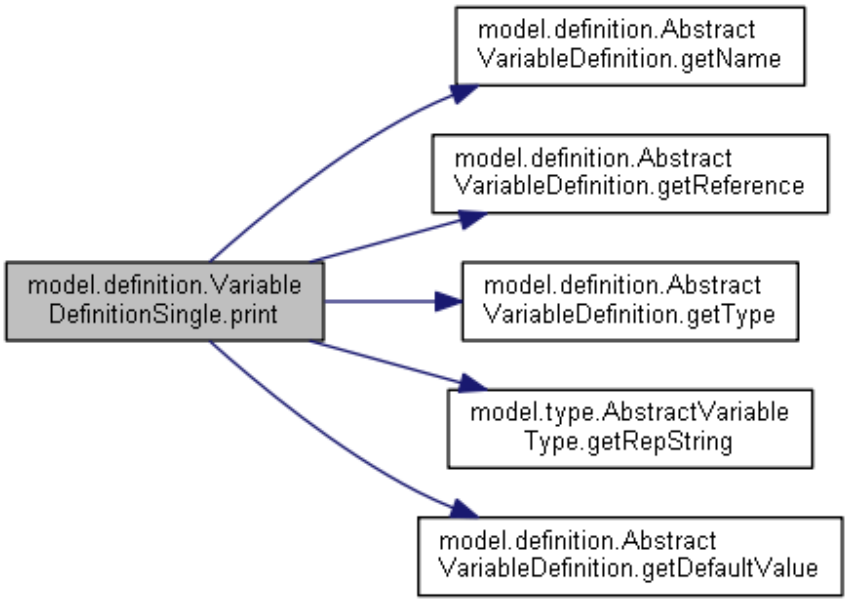
Returns the line to print.

Rückgabe:

Returns a string value, representing a line to print.

Implementiert model.definition.AbstractVariableDefinition.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

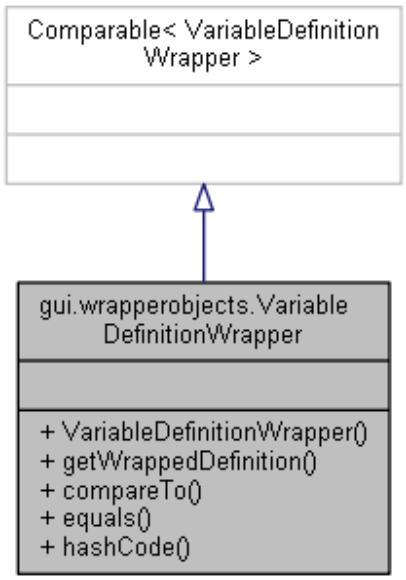


Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

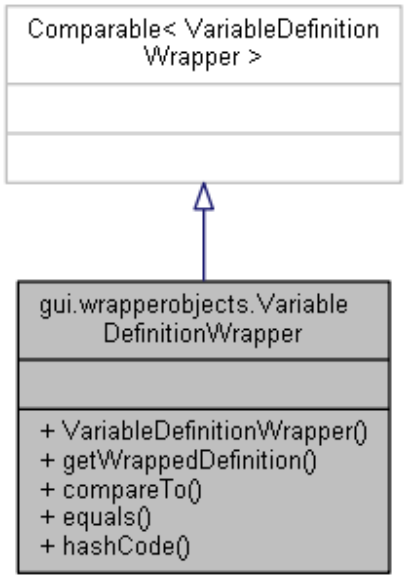
- workspace/ipfli4l13/src/model/definition/VariableDefinitionSingle.java

gui.wrapperobjects.VariableDefinitionWrapper Klassenreferenz

gui.wrapperobjects.VariableDefinitionWrapperKlassendiagramm für gui.wrapperobjects.VariableDefinitionWrapper:



Zusammengehörigkeiten von gui.wrapperobjects.VariableDefinitionWrapper:



Öffentliche Methoden

- VariableDefinitionWrapper (final AbstractVariableDefinition toWrap)
- AbstractVariableDefinition getWrappedDefinition ()

- int **compareTo** (final VariableDefinitionWrapper o)
- boolean **equals** (final Object o)
- int **hashCode** ()

Ausführliche Beschreibung

This class wraps a VariableAssignment.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.wrapperobjects.VariableDefinitionWrapper.VariableDefinitionWrapper (final AbstractVariableDefinition *toWrap*)

Constructor, just sets fields. No side-effects.

Parameter:

toWrap	The assignment to wrap.

Dokumentation der Elementfunktionen

AbstractVariableDefinition gui.wrapperobjects.VariableDefinitionWrapper.getWrappedDefinition ()

Rückgabe:

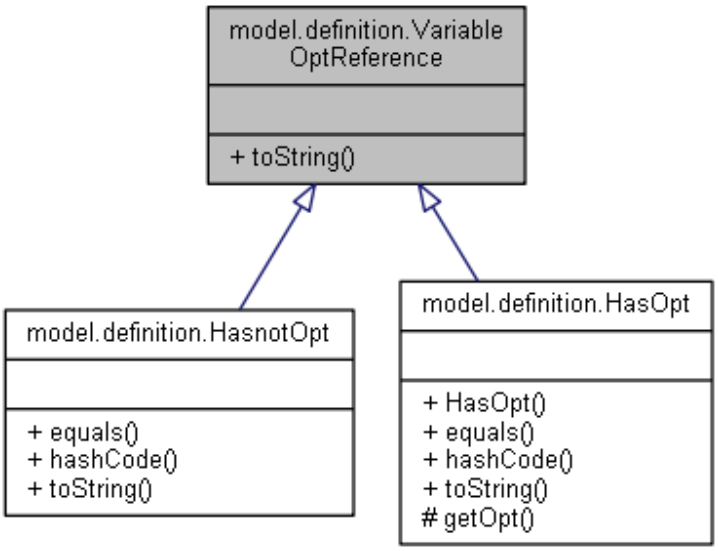
the wrappedAssignment

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

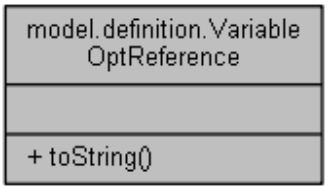
- workspace/ipfli4l13/src/gui/wrapperobjects/VariableDefinitionWrapper.java

model.definition.VariableOptReference Schnittstellenreferenz

model.definition.VariableOptReferenceKlassendiagramm für model.definition.VariableOptReference:



Zusammengehörigkeiten von model.definition.VariableOptReference:



Öffentliche Methoden

- String **toString** ()

Ausführliche Beschreibung

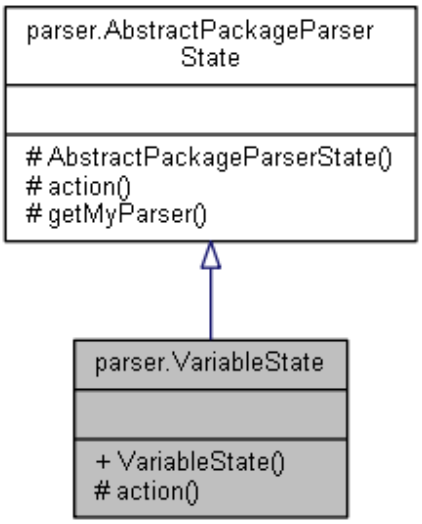
Autor:
HFW410RA - Philipp Rammos

Die Dokumentation für diese Schnittstelle wurde erzeugt aufgrund der Datei:

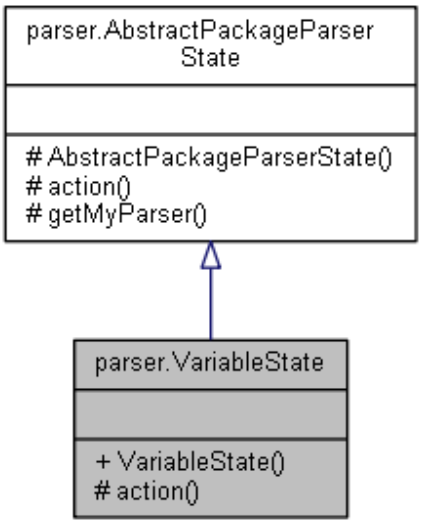
- workspace/ipfli4l13/src/model/definition/VariableOptReference.java

parser.VariableState Klassenreferenz

parser.VariableStateKlassendiagramm für parser.VariableState:



Zusammengehörigkeiten von parser.VariableState:



Öffentliche Methoden

- VariableState (final PackageDescriptionFileParser parser)

Geschützte Methoden

- AbstractPackageParserState **action** (final PackageDescriptionFileSymbol symbol, final Map< String, AbstractVariableDefinition > variableDef) throws ParserException, InterruptedException

Ausführliche Beschreibung

This class represent the VariableState for the Scanner.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

parser.VariableState.VariableState (final PackageDescriptionFileParser *parser*)

This constructor initialize the variable State.

Parameter:

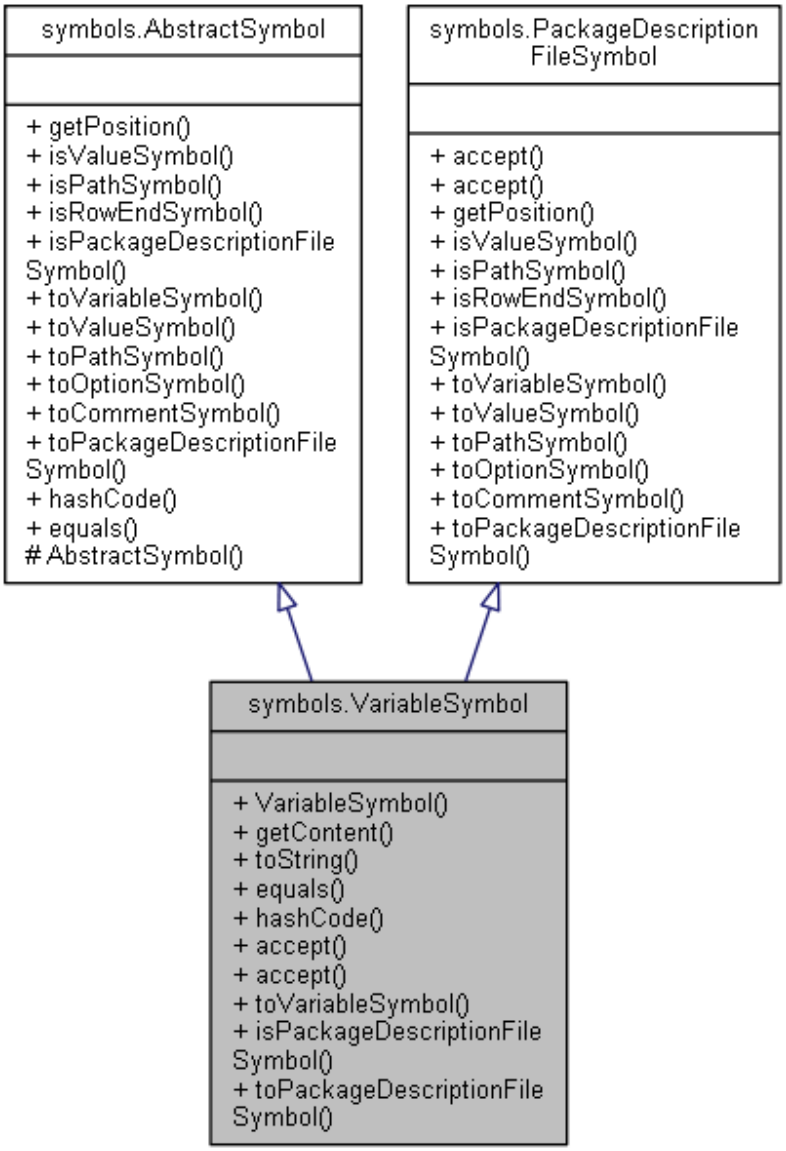
parser	is the packageDescriptionFileParser

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

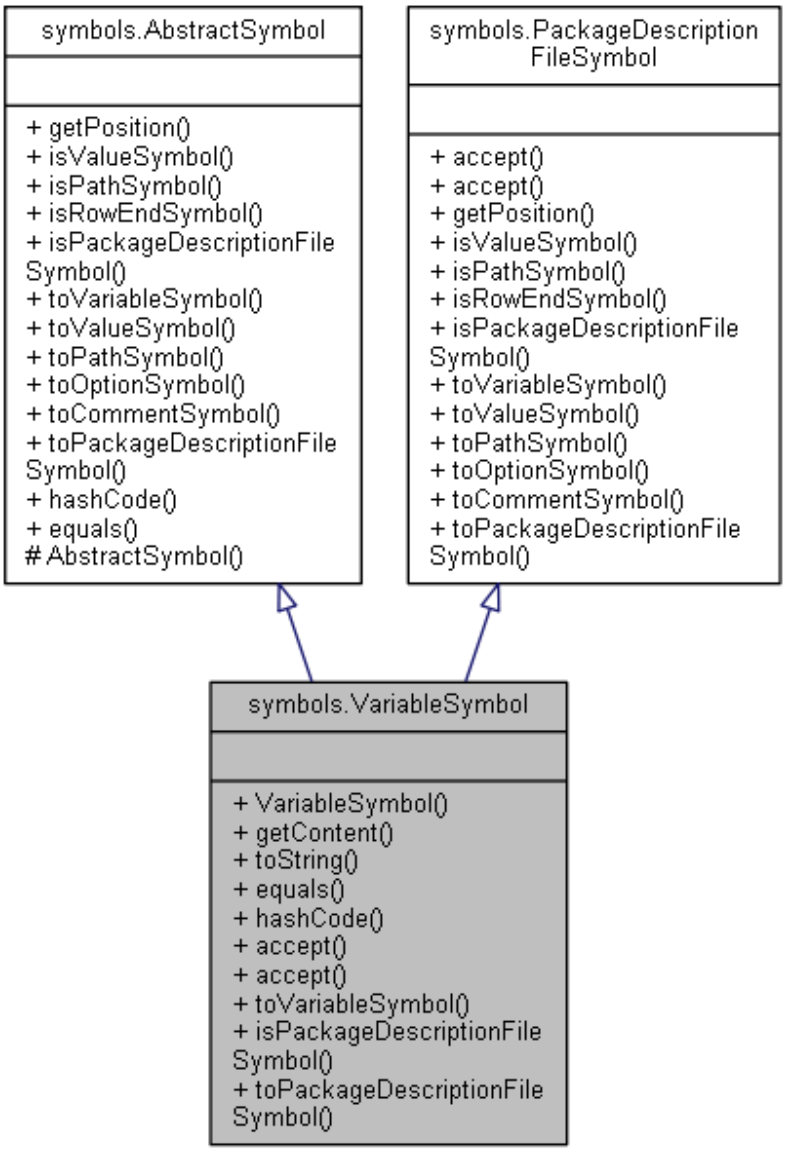
- workspace/ipfli4l13/src/parser/VariableState.java

symbols.VariableSymbol Klassenreferenz

symbols.VariableSymbolKlassendiagramm für symbols.VariableSymbol:



Zusammengehörigkeiten von symbols.VariableSymbol:



Öffentliche Methoden

- VariableSymbol (final String content, final Position position)
- String getContent ()
- String **toString** ()
- boolean **equals** (final Object obj)
- int **hashCode** ()
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitor visitor, final PackageDescriptionFileParser parser) throws InterruptedException, ParserException
- AbstractPackageParserState accept (final PackageDescriptionSymbolVisitorWithTempRule visitor, final PackageDescriptionFileParser parser, final TempPackageRule tempPackageRule) throws InterruptedException, ParserException
- VariableSymbol toVariableSymbol ()
- Boolean isPackageDescriptionFileSymbol ()
- PackageDescriptionFileSymbol toPackageDescriptionFileSymbol ()

Weitere Geerbte Elemente

Ausführliche Beschreibung

This class represents the variable symbol.

Beschreibung der Konstruktoren und Destruktoren

symbols.VariableSymbol.VariableSymbol (final String *content*, final Position *position*)

This constructor instantiate the variable symbol.

Parameter:

position	is the position of the data in the stream.
content	is the content of the VariableState.

Dokumentation der Elementfunktionen

AbstractPackageParserState symbols.VariableSymbol.accept (final PackageDescriptionSymbolVisitor *visitor*, final PackageDescriptionFileParser *parser*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

AbstractPackageParserState symbols.VariableSymbol.accept (final PackageDescriptionSymbolVisitorWithTempRule *visitor*, final PackageDescriptionFileParser *parser*, final TempPackageRule *tempPackageRule*) throws InterruptedException, ParserException

calls the visitor with the Symbollist.

Parameter:

visitor	is the called visitor.
parser	This is the PackageDescriptionparser.
tempPackageRule	is a temporary package rule.

Ausnahmebehandlung:

InterruptedException	from the Buffer.
ParserException	if the symbol is not expected.

Rückgabe:

the actual state of the parser.

Implementiert symbols.PackageDescriptionFileSymbol.

String symbols.VariableSymbol.getContent ()

Rückgabe:

the content of the VariableSymbol.

Boolean symbols.VariableSymbol.isPackageDescriptionFileSymbol ()

Rückgabe:

true, if the Symbol is a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

PackageDescriptionFileSymbol symbols.VariableSymbol.toPackageDescriptionFileSymbol ()

Rückgabe:

a PackageDescriptionFileSymbol or an error, if the symbol is not a package description file symbol.

Implementiert symbols.PackageDescriptionFileSymbol.

VariableSymbol symbols.VariableSymbol.toVariableSymbol ()

Rückgabe:

an error, if the symbol is not a variable symbol

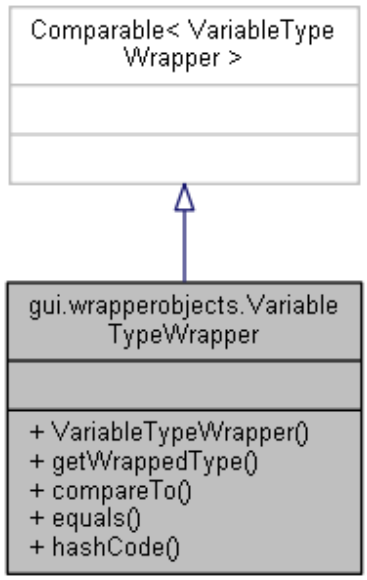
Implementiert symbols.PackageDescriptionFileSymbol.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

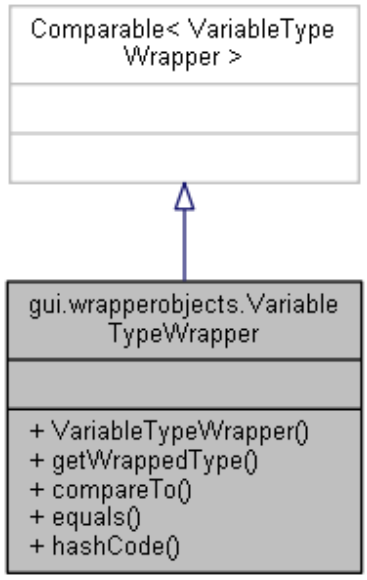
- workspace/ipfli4l13/src/symbols/VariableSymbol.java

gui.wrapperobjects.VariableTypeWrapper Klassenreferenz

gui.wrapperobjects.VariableTypeWrapperKlassendiagramm für gui.wrapperobjects.VariableTypeWrapper:



Zusammengehörigkeiten von gui.wrapperobjects.VariableTypeWrapper:



Öffentliche Methoden

- VariableTypeWrapper (final NamedVariableType toWrap)
- NamedVariableType getWrappedType ()

- int **compareTo** (final VariableTypeWrapper o)
- boolean **equals** (final Object o)
- int **hashCode** ()

Ausführliche Beschreibung

This class wraps a VariableAssignment.

Autor:

Phil

Beschreibung der Konstruktoren und Destruktoren

gui.wrapperobjects.VariableTypeWrapper.VariableTypeWrapper (final NamedVariableType *toWrap*)

Constructor, just sets fields. No side-effects.

Parameter:

toWrap	The assignment to wrap.

Dokumentation der Elementfunktionen

NamedVariableType gui.wrapperobjects.VariableTypeWrapper.getWrappedType ()

Rückgabe:

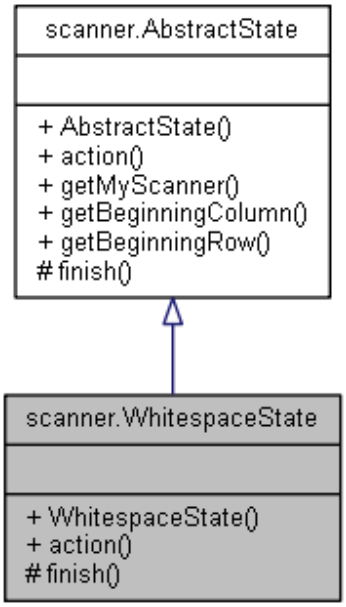
the wrappedAssignment

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

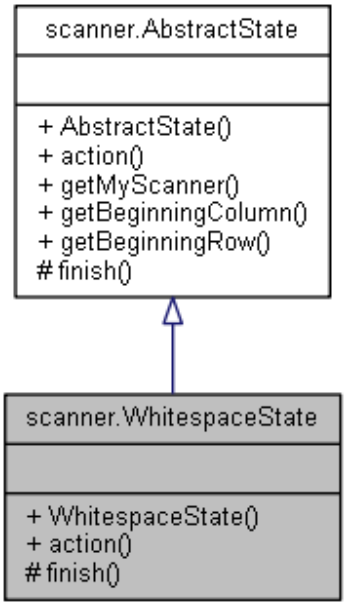
- workspace/ipfli4l13/src/gui/wrapperobjects/VariableTypeWrapper.java

scanner.WhitespaceState Klassenreferenz

scanner.WhitespaceStateKlassendiagramm für scanner.WhitespaceState:



Zusammengehörigkeiten von scanner.WhitespaceState:



Öffentliche Methoden

- WhitespaceState (final Scanner myScanner, final Integer beginningColumn, final Integer beginningRow)

- AbstractState **action** (final Character character, final Buffer< AbstractSymbol > currentResult, final String dataPath)

Geschützte Methoden

- void **finish** (final Buffer< AbstractSymbol > currentResult, final String dataPath)

Ausführliche Beschreibung

This class represents the whitespaceState.

Autor:

HFW410

Beschreibung der Konstruktoren und Destruktoren

scanner.WhitespaceState.WhitespaceState (final Scanner *myScanner*, final Integer *beginningColumn*, final Integer *beginningRow*)

This is the constructor for a new WhitespaceState.

Parameter:

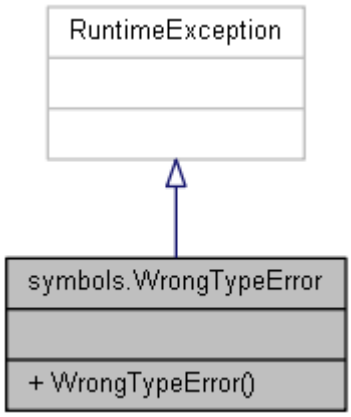
myScanner	is the scanner.
beginningColumn	is the Begin Column of the whitespace
beginningRow	is the beginning row of the symbol

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

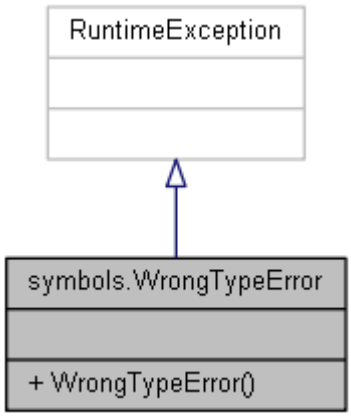
- workspace/ipfli4l13/src/scanner/WhitespaceState.java

symbols.WrongTypeError Klassenreferenz

symbols.WrongTypeErrorKlassendiagramm für symbols.WrongTypeError:



Zusammengehörigkeiten von symbols.WrongTypeError:



Öffentliche Methoden

- WrongTypeError (final String errorMessage)

Ausführliche Beschreibung

is a error, which will be thrown, if the toType-Method will be called with a wrong Type.

Autor:

Lisa Leitloff

Beschreibung der Konstruktoren und Destruktoren

symbols.WrongTypeError.WrongTypeError (final String *errorMessage*)

creates a WrongTypeError.

Parameter:

errorMessage	is the message for the error.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- workspace/ipfli4l13/src/symbols/WrongTypeError.java